Package: ggmlR (via r-universe)

October 18, 2026

Type Package

Title 'GGML' Tensor Operations for Machine Learning

Version 0.8.8

Description Provides 'R' bindings to the 'GGML' tensor library for machine learning, optimized for 'Vulkan' GPU acceleration with a transparent CPU fallback. The package features a 'Keras'-like sequential API and a 'PyTorch'-style 'autograd' engine for building, training, and deploying neural networks. Key capabilities include high-performance 5D tensor operations, 'f16' precision, and efficient quantization. It supports native 'ONNX' model import (50+ operators) and 'GGUF' weight loading from the 'llama.cpp' and 'Hugging Face' ecosystems. Designed for zero-overhead inference via dedicated weight buffering, it integrates seamlessly as a 'parsnip' engine for 'tidymodels' and provides first-class learners for the 'mlr3' framework. See <https://github.com/ggml-org/ggml> for more information about the underlying library.

Depends R (>= 4.1.0)

Imports generics, R6, methods, stats, utils

License MIT + file LICENSE

URL https://github.com/Zabis13/ggmlR

BugReports https://github.com/Zabis13/ggmlR/issues

Encoding UTF-8

SystemRequirements C++17, GNU make, libvulkan-dev, glslc (optional, for GPU on Linux), 'Vulkan' 'SDK' (optional, for GPU on Windows)

Suggests testthat (>= 3.0.0), mlr3 (>= 0.21.0), paradox, digest, parsnip, tibble, rlang, dials, lgr, knitr, rmarkdown, Matrix, Seurat, SeuratObject, RSpectra, irlba, uwot, FNN, SingleCellExperiment, SummarizedExperiment, S4Vectors, withr, hardhat, rsample, tune, workflows

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/build/clean-inst-doc false

Config/pak/sysreqs make

Repository https://zabis13.r-universe.dev

Date/Publication 2026-10-18 07:16:19 UTC

RemoteUrl https://github.com/zabis13/ggmlr

RemoteRef HEAD

RemoteSha 61c835098454641a78c8bec83a07ff06fbdc8a79

Contents

ag_add . . . 17
ag_argmax . . . 18
ag_batch_norm . . . 18
ag_buffer . . . 19
ag_capture . . . 21
ag_capture_free . . . 22
ag_capture_set . . . 22
ag_capture_step . . . 23
ag_checkpoint . . . 24
ag_clamp . . . 25
ag_concat_rows . . . 26
ag_cross_entropy_loss . . . 27
ag_dataloader . . . 27
ag_default_device . . . 28
ag_default_dtype . . . 28
ag_device . . . 29
ag_dropout . . . 29
ag_dtype . . . 30
ag_embedding . . . 30
ag_estimate_training_memory . . . 31
ag_eval . . . 32
ag_exp . . . 33
ag_flash_attention . . . 33
ag_gather . . . 34
ag_get_rows . . . 35
ag_grad . . . 36
ag_gradcheck . . . 36
ag_graph_mode . . . 37
ag_layer_norm . . . 39
ag_linear . . . 40
ag_load_model . . . 41
ag_load_state_dict . . . 42
ag_local_mode . . . 43

ag_log . . . 43
ag_log_softmax . . . 44
ag_matmul . . . 45
ag_matmul_precision . . . 45
ag_max . . . 46
ag_mean . . . 47
ag_mse_loss . . . 47
ag_mul . . . 48
ag_multihead_attention . . . 48
ag_param . . . 50
ag_pow . . . 51
ag_relu . . . 51
ag_reshape . . . 52
ag_sample_categorical . . . 52
ag_sample_normal . . . 53
ag_save_model . . . 54
ag_scale . . . 55
ag_sequential . . . 55
ag_sigmoid . . . 56
ag_slice_rows . . . 56
ag_softmax . . . 57
ag_softmax_cross_entropy_loss . . . 57
ag_state_dict . . . 58
ag_sub . . . 59
ag_sum . . . 59
ag_tanh . . . 60
ag_tape_memory . . . 60
ag_tensor . . . 61
ag_to_device . . . 62
ag_train . . . 62
ag_transpose . . . 63
as.matrix.ag_tensor . . . 63
as.matrix.ggml_matrix . . . 64
as_gpu_matrix . . . 64
augment.ggmlr_parsnip_model . . . 65
backward . . . 66
check_grad_anomaly . . . 66
clip_grad_norm . . . 67
clip_grad_value . . . 68
compile.ggml_sequential_model . . . 69
dequantize_row_iq2_xxs . . . 70
dequantize_row_mxfp4 . . . 71
dequantize_row_nvfp4 . . . 72
dequantize_row_q1_0 . . . 73
dequantize_row_q2_K . . . 73
dequantize_row_q4_0 . . . 74
dequantize_row_tq1_0 . . . 75
dp_train . . . 76

evaluate . . . 77
fit.ggml_sequential_model . . . 78
ggml_abort_is_r_enabled . . . 80
ggml_abs . . . 80
ggml_abs_inplace . . . 81
ggml_add . . . 82
ggml_add_inplace . . . 83
ggml_add_rel_pos . . . 84
ggml_add1 . . . 84
ggml_apply . . . 85
ggml_arange . . . 86
ggml_are_same_layout . . . 86
ggml_are_same_shape . . . 87
ggml_are_same_stride . . . 87
ggml_argmax . . . 88
ggml_argsort . . . 89
ggml_attention . . . 89
ggml_backend_alloc_ctx_tensors . . . 91
ggml_backend_buffer_clear . . . 92
ggml_backend_buffer_free . . . 93
ggml_backend_buffer_get_size . . . 93
ggml_backend_buffer_get_usage . . . 94
ggml_backend_buffer_is_host . . . 95
ggml_backend_buffer_is_multi_buffer . . . 96
ggml_backend_buffer_name . . . 97
ggml_backend_buffer_reset . . . 97
ggml_backend_buffer_set_usage . . . 98
ggml_backend_buffer_usage_any . . . 99
ggml_backend_buffer_usage_compute . . . 100
ggml_backend_buffer_usage_weights . . . 101
ggml_backend_buft_get_alignment . . . 102
ggml_backend_buft_get_max_size . . . 103
ggml_backend_buft_is_host . . . 104
ggml_backend_buft_name . . . 105
ggml_backend_cpu_init . . . 106
ggml_backend_cpu_set_n_threads . . . 106
ggml_backend_dev_buffer_type . . . 107
ggml_backend_dev_by_name . . . 108
ggml_backend_dev_by_type . . . 109
ggml_backend_dev_count . . . 110
ggml_backend_dev_description . . . 110
ggml_backend_dev_get . . . 111
ggml_backend_dev_get_props . . . 112
ggml_backend_dev_host_buffer_type . . . 113
ggml_backend_dev_init . . . 114
ggml_backend_dev_memory . . . 115
ggml_backend_dev_name . . . 116
ggml_backend_dev_offload_op . . . 117

ggml_backend_dev_supports_buft . . . 118
ggml_backend_dev_supports_op . . . 119
ggml_backend_dev_type . . . 120
ggml_backend_device_register . . . 121
ggml_backend_device_type_accel . . . 122
ggml_backend_device_type_cpu . . . 122
ggml_backend_device_type_gpu . . . 123
ggml_backend_device_type_igpu . . . 124
ggml_backend_event_free . . . 125
ggml_backend_event_new . . . 126
ggml_backend_event_record . . . 127
ggml_backend_event_synchronize . . . 128
ggml_backend_event_wait . . . 129
ggml_backend_free . . . 130
ggml_backend_get_device . . . 130
ggml_backend_graph_compute . . . 131
ggml_backend_graph_compute_async . . . 132
ggml_backend_graph_plan_compute . . . 133
ggml_backend_graph_plan_create . . . 134
ggml_backend_graph_plan_free . . . 135
ggml_backend_init_best . . . 136
ggml_backend_init_by_name . . . 137
ggml_backend_init_by_type . . . 138
ggml_backend_load . . . 139
ggml_backend_load_all . . . 140
ggml_backend_meta_device . . . 140
ggml_backend_multi_buffer_alloc_buffer . . . 142
ggml_backend_multi_buffer_set_usage . . . 143
ggml_backend_name . . . 144
ggml_backend_reg_by_name . . . 145
ggml_backend_reg_count . . . 146
ggml_backend_reg_dev_count . . . 146
ggml_backend_reg_dev_get . . . 147
ggml_backend_reg_get . . . 148
ggml_backend_reg_name . . . 149
ggml_backend_register . . . 150
ggml_backend_sched_alloc_graph . . . 151
ggml_backend_sched_free . . . 152
ggml_backend_sched_get_backend . . . 152
ggml_backend_sched_get_n_backends . . . 153
ggml_backend_sched_get_n_copies . . . 153
ggml_backend_sched_get_n_splits . . . 154
ggml_backend_sched_get_tensor_backend . . . 154
ggml_backend_sched_graph_compute . . . 155
ggml_backend_sched_graph_compute_async . . . 156
ggml_backend_sched_new . . . 156
ggml_backend_sched_reserve . . . 157
ggml_backend_sched_reset . . . 158

ggml_backend_sched_set_tensor_backend . . . 159
ggml_backend_sched_synchronize . . . 159
ggml_backend_sched_trace . . . 160
ggml_backend_synchronize . . . 160
ggml_backend_tensor_copy_async . . . 161
ggml_backend_tensor_get_and_sync . . . 162
ggml_backend_tensor_get_async . . . 163
ggml_backend_tensor_get_data . . . 164
ggml_backend_tensor_get_f32_first . . . 164
ggml_backend_tensor_set_async . . . 165
ggml_backend_tensor_set_data . . . 166
ggml_backend_unload . . . 166
ggml_batch_norm . . . 167
ggml_blck_size . . . 168
ggml_build_backward_expand . . . 168
ggml_build_forward_expand . . . 169
ggml_build_forward_expand_grads . . . 170
ggml_callback_early_stopping . . . 172
ggml_can_repeat . . . 172
ggml_cast . . . 173
ggml_cast_numeric . . . 174
ggml_ceil . . . 175
ggml_ceil_inplace . . . 175
ggml_clamp . . . 176
ggml_compile.ggml_functional_model . . . 177
ggml_concat . . . 179
ggml_cont . . . 179
ggml_conv_1d . . . 180
ggml_conv_1d_dw . . . 181
ggml_conv_1d_layer . . . 181
ggml_conv_2d . . . 182
ggml_conv_2d_direct . . . 183
ggml_conv_2d_dw . . . 184
ggml_conv_2d_dw_direct . . . 185
ggml_conv_2d_layer . . . 185
ggml_conv_transpose_1d . . . 187
ggml_conv_transpose_2d_p0 . . . 187
ggml_cos . . . 188
ggml_count_equal . . . 188
ggml_cpu_add . . . 189
ggml_cpu_features . . . 190
ggml_cpu_get_rvv_vlen . . . 191
ggml_cpu_get_sve_cnt . . . 191
ggml_cpu_has_amx_int8 . . . 192
ggml_cpu_has_arm_fma . . . 192
ggml_cpu_has_avx . . . 193
ggml_cpu_has_avx_vnni . . . 194
ggml_cpu_has_avx2 . . . 194

ggml_cpu_has_avx512 . . . 195
ggml_cpu_has_avx512_bf16 . . . 195
ggml_cpu_has_avx512_vbmi . . . 196
ggml_cpu_has_avx512_vnni . . . 197
ggml_cpu_has_bmi2 . . . 197
ggml_cpu_has_dotprod . . . 198
ggml_cpu_has_f16c . . . 198
ggml_cpu_has_fma . . . 199
ggml_cpu_has_fp16_va . . . 200
ggml_cpu_has_llamafile . . . 200
ggml_cpu_has_matmul_int8 . . . 201
ggml_cpu_has_neon . . . 201
ggml_cpu_has_riscv_v . . . 202
ggml_cpu_has_sme . . . 203
ggml_cpu_has_sse3 . . . 203
ggml_cpu_has_ssse3 . . . 204
ggml_cpu_has_sve . . . 205
ggml_cpu_has_vsx . . . 205
ggml_cpu_has_vxe . . . 206
ggml_cpu_has_wasm_simd . . . 206
ggml_cpu_mul . . . 207
ggml_cpy . . . 208
ggml_cross_entropy_loss . . . 208
ggml_cross_entropy_loss_back . . . 209
ggml_crossprod . . . 210
ggml_cumsum . . . 210
ggml_custom . . . 211
ggml_custom_inplace . . . 212
ggml_custom_ops . . . 213
ggml_cycles . . . 214
ggml_cycles_per_ms . . . 215
ggml_default_mlp . . . 215
ggml_dense . . . 217
ggml_diag . . . 218
ggml_diag_mask_inf . . . 218
ggml_diag_mask_inf_inplace . . . 219
ggml_diag_mask_zero . . . 220
ggml_div . . . 220
ggml_div_inplace . . . 221
ggml_dup . . . 221
ggml_dup_inplace . . . 222
ggml_dup_tensor . . . 223
ggml_element_size . . . 223
ggml_elu . . . 224
ggml_elu_inplace . . . 224
ggml_embedding . . . 225
ggml_estimate_memory . . . 225
ggml_evaluate.ggml_functional_model . . . 226

ggml_exp . . . 228
ggml_exp_inplace . . . 228
ggml_extract . . . 229
ggml_fit.ggml_functional_model . . . 230
ggml_fit_generator . . . 233
ggml_fit_opt . . . 235
ggml_fit_opt_gen . . . 237
ggml_fit_opt_multi . . . 239
ggml_flash_attn_back . . . 242
ggml_flash_attn_ext . . . 243
ggml_flash_attn_ext_add_sinks . . . 244
ggml_flash_attn_ext_get_prec . . . 245
ggml_flash_attn_ext_set_prec . . . 245
ggml_floor . . . 246
ggml_floor_inplace . . . 246
ggml_free . . . 247
ggml_freeze_bn_stats . . . 247
ggml_freeze_weights . . . 248
ggml_ftype_to_ggml_type . . . 250
ggml_gallocr_alloc_graph . . . 250
ggml_gallocr_free . . . 251
ggml_gallocr_get_buffer_size . . . 251
ggml_gallocr_new . . . 252
ggml_gallocr_new_buft . . . 253
ggml_gallocr_reserve . . . 253
ggml_gated_linear_attn . . . 254
ggml_gated_linear_attn_back . . . 255
ggml_geglu . . . 255
ggml_geglu_quick . . . 256
ggml_geglu_split . . . 257
ggml_gelu . . . 257
ggml_gelu_back . . . 258
ggml_gelu_erf . . . 259
ggml_gelu_inplace . . . 259
ggml_gelu_quick . . . 260
ggml_get_f32 . . . 261
ggml_get_f32_nd . . . 262
ggml_get_first_tensor . . . 262
ggml_get_i32 . . . 263
ggml_get_i32_nd . . . 263
ggml_get_layer . . . 264
ggml_get_max_tensor_size . . . 264
ggml_get_mem_size . . . 265
ggml_get_n_threads . . . 266
ggml_get_name . . . 266
ggml_get_next_tensor . . . 267
ggml_get_no_alloc . . . 267
ggml_get_op_params . . . 268

ggml_get_op_params_f32 . . . 268
ggml_get_op_params_i32 . . . 269
ggml_get_rel_pos . . . 269
ggml_get_rows . . . 270
ggml_get_rows_back . . . 271
ggml_get_unary_op . . . 271
ggml_glu . . . 272
GGML_GLU_OP_REGLU . . . 273
ggml_glu_split . . . 274
ggml_graph_add_node . . . 274
ggml_graph_clear . . . 275
ggml_graph_compute . . . 275
ggml_graph_compute_with_ctx . . . 276
ggml_graph_cpy . . . 277
ggml_graph_dump_dot . . . 278
ggml_graph_dup . . . 278
ggml_graph_expand . . . 279
ggml_graph_get_grad . . . 280
ggml_graph_get_tensor . . . 281
ggml_graph_n_nodes . . . 281
ggml_graph_node . . . 282
ggml_graph_overhead . . . 282
ggml_graph_print . . . 283
ggml_graph_reset . . . 283
ggml_graph_view . . . 284
ggml_group_norm . . . 285
ggml_group_norm_inplace . . . 285
ggml_gru . . . 286
ggml_hardsigmoid . . . 287
ggml_hardswish . . . 288
ggml_im2col . . . 288
ggml_init . . . 289
ggml_init_auto . . . 290
ggml_inject . . . 291
ggml_input . . . 292
ggml_is_available . . . 293
ggml_is_contiguous . . . 293
ggml_is_contiguous_0 . . . 294
ggml_is_contiguous_1 . . . 294
ggml_is_contiguous_2 . . . 295
ggml_is_contiguous_channels . . . 295
ggml_is_contiguous_rows . . . 296
ggml_is_contiguously_allocated . . . 296
ggml_is_permuted . . . 297
ggml_is_quantized . . . 298
ggml_is_transposed . . . 298
ggml_l2_norm . . . 299
ggml_l2_norm_inplace . . . 300

ggml_layer_add . . . 300
ggml_layer_attention . . . 301
ggml_layer_batch_norm . . . 303
ggml_layer_concatenate . . . 304
ggml_layer_conv_1d . . . 305
ggml_layer_conv_2d . . . 306
ggml_layer_custom . . . 307
ggml_layer_dense . . . 308
ggml_layer_dropout . . . 309
ggml_layer_embedding . . . 311
ggml_layer_flatten . . . 312
ggml_layer_global_average_pooling_2d . . . 313
ggml_layer_global_max_pooling_2d . . . 313
ggml_layer_gru . . . 314
ggml_layer_layer_norm . . . 315
ggml_layer_lstm . . . 316
ggml_layer_max_pooling_2d . . . 317
ggml_layer_permute . . . 318
ggml_layer_positional_embedding . . . 319
ggml_layer_reshape . . . 320
ggml_layer_rms_norm . . . 321
ggml_layer_sequence_pooling . . . 322
ggml_layer_transformer_block . . . 323
ggml_leaky_relu . . . 325
ggml_load_model . . . 326
ggml_load_weights . . . 326
ggml_log . . . 327
ggml_log_inplace . . . 328
ggml_log_is_r_enabled . . . 328
ggml_log_set_default . . . 329
ggml_log_set_r . . . 329
ggml_lstm . . . 330
ggml_marshal_model . . . 330
ggml_matmul . . . 331
ggml_matmul_f64 . . . 332
ggml_matrix-class . . . 333
ggml_mean . . . 334
ggml_model . . . 335
ggml_model_backend . . . 336
ggml_model_sequential . . . 336
ggml_mul . . . 337
ggml_mul_inplace . . . 338
ggml_mul_mat . . . 338
ggml_mul_mat_id . . . 339
ggml_n_dims . . . 340
GGML_N_TASKS_MAX . . . 341
ggml_nbytes . . . 341
ggml_neg . . . 342

ggml_neg_inplace . . . 343
ggml_nelements . . . 343
ggml_new_f32 . . . 344
ggml_new_i32 . . . 345
ggml_new_tensor . . . 345
ggml_new_tensor_1d . . . 346
ggml_new_tensor_2d . . . 347
ggml_new_tensor_3d . . . 347
ggml_new_tensor_4d . . . 348
ggml_norm . . . 349
ggml_norm_back . . . 350
ggml_norm_inplace . . . 350
ggml_nrows . . . 351
ggml_op_can_inplace . . . 351
ggml_op_desc . . . 352
ggml_op_name . . . 353
ggml_op_symbol . . . 353
ggml_ops_registry . . . 354
ggml_opt_alloc . . . 354
ggml_opt_context_optimizer_type . . . 355
ggml_opt_dataset_data . . . 356
ggml_opt_dataset_free . . . 357
ggml_opt_dataset_get_batch . . . 357
ggml_opt_dataset_get_batch_head . . . 358
ggml_opt_dataset_init . . . 360
ggml_opt_dataset_labels . . . 361
ggml_opt_dataset_ndata . . . 362
ggml_opt_dataset_shuffle . . . 363
ggml_opt_dataset_weights . . . 364
ggml_opt_default_params . . . 365
ggml_opt_epoch . . . 366
ggml_opt_eval . . . 367
ggml_opt_fit . . . 368
ggml_opt_fit_multi . . . 369
ggml_opt_free . . . 371
ggml_opt_get_lr . . . 372
ggml_opt_grad_acc . . . 373
ggml_opt_init . . . 374
ggml_opt_init_for_fit . . . 375
ggml_opt_init_for_fit_multi . . . 376
ggml_opt_inputs . . . 377
ggml_opt_labels . . . 378
ggml_opt_loss . . . 379
ggml_opt_loss_i . . . 380
ggml_opt_loss_type_binary_cross_entropy . . . 381
ggml_opt_loss_type_cross_entropy . . . 382
ggml_opt_loss_type_huber . . . 382
ggml_opt_loss_type_mae . . . 383

ggml_opt_loss_type_mean . . . 384
ggml_opt_loss_type_mse . . . 385
ggml_opt_loss_type_sum . . . 386
ggml_opt_loss_type_weighted_mse . . . 386
ggml_opt_n_loss . . . 387
ggml_opt_ncorrect . . . 388
ggml_opt_optimizer_name . . . 389
ggml_opt_optimizer_type_adamw . . . 390
ggml_opt_optimizer_type_sgd . . . 390
ggml_opt_outputs . . . 391
ggml_opt_outputs_i . . . 392
ggml_opt_pred . . . 393
ggml_opt_prepare_alloc . . . 394
ggml_opt_reset . . . 395
ggml_opt_result_accuracy . . . 396
ggml_opt_result_free . . . 397
ggml_opt_result_init . . . 397
ggml_opt_result_loss . . . 398
ggml_opt_result_loss_i . . . 399
ggml_opt_result_n_loss . . . 400
ggml_opt_result_ndata . . . 401
ggml_opt_result_pred . . . 402
ggml_opt_result_reset . . . 402
ggml_opt_set_lr . . . 403
ggml_opt_static_graphs . . . 404
ggml_opt_step_adamw . . . 405
ggml_out_prod . . . 406
ggml_pad . . . 407
ggml_pad_ext . . . 408
ggml_pad_reflect_1d . . . 408
ggml_permute . . . 409
ggml_pool_1d . . . 409
ggml_pool_2d . . . 411
ggml_pop_layer . . . 411
ggml_pp_dp_forward . . . 412
ggml_pp_forward . . . 413
ggml_predict.ggml_functional_model . . . 415
ggml_predict_classes . . . 416
ggml_print_mem_status . . . 416
ggml_print_objects . . . 417
ggml_quant_block_info . . . 417
ggml_quantize_chunk . . . 418
ggml_quantize_free . . . 419
ggml_quantize_init . . . 419
ggml_quantize_requires_imatrix . . . 420
ggml_reglu . . . 420
ggml_reglu_split . . . 421
ggml_relu . . . 422

ggml_relu_inplace . . . 422
ggml_repeat . . . 423
ggml_repeat_back . . . 424
ggml_reset . . . 424
ggml_reshape_1d . . . 425
ggml_reshape_2d . . . 426
ggml_reshape_3d . . . 426
ggml_reshape_4d . . . 427
ggml_result . . . 428
ggml_rms_norm . . . 428
ggml_rms_norm_back . . . 429
ggml_rms_norm_inplace . . . 430
ggml_roll . . . 430
ggml_rope . . . 431
ggml_rope_ext . . . 432
ggml_rope_ext_back . . . 434
ggml_rope_ext_inplace . . . 435
ggml_rope_inplace . . . 436
ggml_rope_multi . . . 436
ggml_rope_multi_back . . . 438
ggml_rope_multi_inplace . . . 439
ggml_round . . . 440
ggml_round_inplace . . . 441
ggml_run . . . 441
ggml_rwkv_output . . . 442
ggml_rwkv_wkv6 . . . 443
ggml_rwkv_wkv6_back . . . 444
ggml_rwkv_wkv7 . . . 445
ggml_rwkv_wkv7_back . . . 446
ggml_save_model . . . 446
ggml_save_weights . . . 447
ggml_scale . . . 448
ggml_scale_bias . . . 449
ggml_scale_inplace . . . 449
ggml_scatter_elements . . . 450
ggml_schedule_cosine_decay . . . 451
ggml_schedule_reduce_on_plateau . . . 451
ggml_schedule_step_decay . . . 452
ggml_set . . . 453
ggml_set_1d . . . 453
ggml_set_2d . . . 454
ggml_set_abort_callback_default . . . 454
ggml_set_abort_callback_r . . . 455
ggml_set_f32 . . . 455
ggml_set_f32_nd . . . 456
ggml_set_i32 . . . 457
ggml_set_i32_nd . . . 457
ggml_set_input . . . 458

ggml_set_loss . . . 458
ggml_set_n_threads . . . 459
ggml_set_name . . . 459
ggml_set_no_alloc . . . 460
ggml_set_omp_threads . . . 460
ggml_set_op_params . . . 461
ggml_set_op_params_f32 . . . 461
ggml_set_op_params_i32 . . . 462
ggml_set_output . . . 463
ggml_set_param . . . 463
ggml_set_seed . . . 464
ggml_set_zero . . . 465
ggml_sgn . . . 465
ggml_sigmoid . . . 466
ggml_sigmoid_inplace . . . 467
ggml_silu . . . 467
ggml_silu_back . . . 468
ggml_silu_inplace . . . 468
ggml_sin . . . 469
ggml_snapshot_free . . . 470
ggml_soft_max . . . 471
ggml_soft_max_add_sinks . . . 471
ggml_soft_max_ext . . . 472
ggml_soft_max_ext_back . . . 473
ggml_soft_max_ext_back_inplace . . . 473
ggml_soft_max_ext_inplace . . . 474
ggml_soft_max_inplace . . . 475
ggml_softplus . . . 475
ggml_softplus_inplace . . . 476
GGML_SORT_ORDER_ASC . . . 476
ggml_sqr . . . 477
ggml_sqr_inplace . . . 478
ggml_sqrt . . . 478
ggml_sqrt_inplace . . . 479
ggml_ssm_conv . . . 480
ggml_ssm_conv_back . . . 481
ggml_ssm_scan . . . 482
ggml_ssm_scan_back . . . 483
ggml_ssm_scan_output . . . 484
ggml_step . . . 485
ggml_sub . . . 486
ggml_sub_inplace . . . 486
ggml_sum . . . 487
ggml_sum_rows . . . 488
ggml_swiglu . . . 488
ggml_swiglu_split . . . 489
ggml_tanh . . . 490
ggml_tanh_inplace . . . 490

ggml_task . . . 491
ggml_tcrossprod . . . 491
ggml_tensor_copy . . . 492
ggml_tensor_data_ptr . . . 492
ggml_tensor_nb . . . 493
ggml_tensor_num . . . 493
ggml_tensor_overhead . . . 494
ggml_tensor_set_f32_scalar . . . 494
ggml_tensor_shape . . . 495
ggml_tensor_type . . . 495
ggml_test . . . 496
ggml_test_adamw_steps . . . 496
ggml_test_backend_ops . . . 497
ggml_time_init . . . 498
ggml_time_ms . . . 499
ggml_time_us . . . 499
ggml_timestep_embedding . . . 500
ggml_top_k . . . 500
ggml_tp_dp_forward . . . 501
ggml_trainer . . . 502
ggml_training_history . . . 506
ggml_transpose . . . 507
GGML_TYPE_F32 . . . 507
ggml_type_name . . . 509
ggml_type_size . . . 510
ggml_type_sizef . . . 510
ggml_unary_op_name . . . 511
ggml_unfreeze_weights . . . 511
ggml_unmarshal_model . . . 512
ggml_upscale . . . 513
ggml_used_mem . . . 514
ggml_version . . . 515
ggml_view_1d . . . 515
ggml_view_2d . . . 516
ggml_view_3d . . . 516
ggml_view_4d . . . 517
ggml_view_tensor . . . 518
ggml_vulkan_available . . . 518
ggml_vulkan_backend_name . . . 519
ggml_vulkan_device_caps . . . 519
ggml_vulkan_device_count . . . 520
ggml_vulkan_device_description . . . 520
ggml_vulkan_device_groups . . . 521
ggml_vulkan_device_memory . . . 521
ggml_vulkan_free . . . 522
ggml_vulkan_hard_exit_available . . . 523
ggml_vulkan_init . . . 523
ggml_vulkan_is_backend . . . 524

ggml_vulkan_list_devices . . . 525
ggml_vulkan_p2p_selftest . . . 525
ggml_vulkan_shutdown . . . 527
ggml_vulkan_split_buffer_type . . . 528
ggml_vulkan_split_mul_mat . . . 529
ggml_vulkan_split_row_ranges . . . 531
ggml_vulkan_status . . . 532
ggml_win_part . . . 532
ggml_win_unpart . . . 533
ggml_with_temp_ctx . . . 534
gguf_free . . . 534
gguf_load . . . 535
gguf_metadata . . . 535
gguf_tensor_data . . . 536
gguf_tensor_info . . . 536
gguf_tensor_names . . . 537
glance.ggmlr_parsnip_model . . . 537
iq2xs_free_impl . . . 538
iq2xs_init_impl . . . 539
iq3xs_free_impl . . . 539
iq3xs_init_impl . . . 540
lr_scheduler_cosine . . . 541
lr_scheduler_cyclic . . . 542
lr_scheduler_onecycle . . . 543
lr_scheduler_step . . . 544
lr_scheduler_warmup_cosine . . . 545
nn_topo_sort . . . 546
onnx_device_info . . . 546
onnx_graph_info . . . 547
onnx_inputs . . . 548
onnx_load . . . 548
onnx_run . . . 549
onnx_summary . . . 550
optimizer_adam . . . 550
optimizer_sgd . . . 551
plot.ggml_history . . . 552
predict.ggml_sequential_model . . . 553
predict.onnx_model . . . 554
print.ag_tensor . . . 555
print.ggml_functional_model . . . 555
print.ggml_history . . . 556
print.ggml_sequential_model . . . 556
print.onnx_model . . . 557
pt_cube4_layout . . . 557
pt_forward . . . 558
pt_transformer . . . 559
pt_transformer_load . . . 560
pth_catalogue . . . 560

pth_load . . . 561
quantize_iq2_xxs . . . 562
quantize_mxfp4 . . . 563
quantize_nvfp4 . . . 564
quantize_q1_0 . . . 565
quantize_q2_K . . . 565
quantize_q4_0 . . . 566
quantize_row_iq3_xxs_ref . . . 567
quantize_row_mxfp4_ref . . . 568
quantize_row_nvfp4_ref . . . 569
quantize_row_q1_0_ref . . . 569
quantize_row_q2_K_ref . . . 570
quantize_row_q4_0_ref . . . 571
quantize_row_tq1_0_ref . . . 572
quantize_tq1_0 . . . 572
rope_types . . . 573
RunGGML.SingleCellExperiment . . . 574
summary.ggml_sequential_model . . . 577
tidy.ggmlr_parsnip_model . . . 577
with_grad_tape . . . 578

Index **580**

---

ag_add *Element-wise addition with broadcasting*

Description

Computes `A + B`. If `B` is `[m, 1]` and `A` is `[m, n]`, `B` is broadcast across columns (useful for bias vectors).

Usage

```
ag_add(A, B)
```

Arguments

`A`	ag_tensor or numeric matrix
`B`	ag_tensor or numeric matrix (may be `[m,1]` or `[1,n]` for broadcasting)

Value

ag_tensor

ag_argmax *Column-wise argmax*

Description

Index of the largest entry in every column of `x`, for action selection over a `[n_actions, batch]` matrix of logits or log-probabilities.

Usage

```
ag_argmax(x)
```

Arguments

`x`	ag_tensor or numeric matrix, `[n, batch]`.

Details

Ties go to the FIRST index on every path (R, CPU kernel, Vulkan shader), matching `which.max`. `NaN` is never selected; a column with nothing above `-Inf` (all `NaN` / all `-Inf`) gives `0`.

Not differentiable. On the GPU this reads the result back to the host, so it is a synchronisation point: fine where the environment needs the action, but keep it out of the training step.

Value

Integer vector of length `ncol(x)`, 0-based.

Examples

```
ag_argmax(matrix(c(1, 3, 3,  5, 2, 5), 3, 2))   # 1 0
```

ag_batch_norm *Create a Batch Normalisation layer*

Description

Normalises each feature (row) over the batch dimension. Learnable scale `gamma` [F,1] and shift `beta` [F,1].

Usage

```
ag_batch_norm(num_features, eps = 1e-05, momentum = 0.1)
```

Arguments

num_features	Number of features (rows of input)
eps	Numerical stability constant (default 1e-5)
momentum	Running-stats momentum (default 0.1)

Details

Training mode: use batch statistics; update running mean/var. **Eval mode**: use stored running statistics.

Value

An ag_batch_norm environment

Examples

```
bn <- ag_batch_norm(16L)
x  <- ag_tensor(matrix(rnorm(16 * 32), 16, 32))
out <- bn$forward(x)
```

---

ag_buffer *Device buffer for column-indexed reads*

Description

A matrix of rows x capacity kept on the compute device without a gradient: written in place by column ranges with ag_buffer_write() and read by column indices with ag_get_rows() inside ag_* computations, including ag_capture() recordings. Typical use is a replay buffer: one column per transition.

Usage

```
ag_buffer(rows, capacity, dtype = "f32")

ag_buffer_write(buf, data, col_offset)

ag_buffer_read(buf, col_offset = 0, k = buf$capacity - col_offset)

ag_buffer_free(buf)
```

Arguments

rows	Number of rows (values per column).
capacity	Number of columns.
dtype	Storage type. Only "f32" is supported.
buf	An ag_buffer.
data	Numeric matrix with rows rows (a vector is one column).
col_offset	0-based index of the first column written or read.
k	Number of columns to read.

Details

On the CPU the buffer is an R matrix and the same calls work on it. On the GPU the memory is allocated when the buffer is created (or first used on the device) and zero-filled; creation fails with an error when the buffer needs more than 85% of the free memory the device reports (driver estimates are approximate). A device switch (ag_device("cpu")) downloads the value to the host in 16 MB parts before the backend is released (the host copy is an R double matrix, about twice the buffer's device size), and the switch fails with an error if that download does; the next use on the GPU uploads it again, and recordings that read the buffer are then made anew.

Writes are synchronous: when ag_buffer_write() returns, the data is on the device, and any graph computed afterwards, including a captured replay, reads it. Graph nodes queued before the write are computed first, so they read the old value.

ag_buffer_write() writes columns col_offset to col_offset + ncol(data) - 1; the range must lie inside the buffer (a ring buffer wraps with two writes).

ag_buffer_read() returns columns col_offset to col_offset + k - 1 as a rows x k matrix; reading a large buffer in parts bounds the host memory used.

ag_buffer_free() releases the memory at once instead of at garbage collection; the buffer cannot be used afterwards.

Value

An object of class ag_buffer with fields rows, capacity and dtype.

See Also

ag_get_rows

Examples

```
buf <- ag_buffer(3, 10)
ag_buffer_write(buf, matrix(1:6, 3, 2), col_offset = 4)
ag_buffer_read(buf, 4, 2)
as.matrix(ag_get_rows(buf, c(5, 4, 5)))
ag_buffer_free(buf)
```

ag_capture *Capture an ag_* computation as a reusable device graph*

Description

Wraps `fn` so that, on the GPU, its ag_* operations are recorded once per input shape and every later call runs the recorded ggml graph: upload the inputs, one graph compute, download the outputs – no R-level op dispatch. On the CPU the wrapper simply calls `fn`, so the same code runs on both.

Usage

```
ag_capture(fn, params = NULL)
```

Arguments

`fn`	Function of named ag_tensor arguments returning an ag_tensor or a named list of ag_tensors. Forward computations only (no `with_grad_tape`).
`params`	Optional list of ag_tensors (typically the model parameters) and `ag_buffer`s read with `ag_get_rows` whose device memory the recording may reference; a change re-records.

Details

`fn` must depend on the data only through its arguments. Anything it computes in R from data, or uploads itself (`ag_tensor(f(x))`), would be frozen into the graph at the first call; such uploads, any download and any eager launch during the recording are errors. Weights must be resident on the device (e.g. trained or used on the GPU) before the first call.

The recording is redone when the input shapes, compute dtype or matmul precision change, when the persistent device pool is reset (device switch), and when a tensor in `params` no longer points at the device memory it had at recording time.

Value

A function taking the same named arguments as numeric matrices (a vector is one column) and returning a matrix, or a named list of matrices.

See Also

`ag_capture_free`

Examples

```
W <- ag_param(matrix(rnorm(6), 2, 3))
f <- ag_capture(function(x) ag_relu(ag_matmul(W, x)), params = list(W))
f(x = matrix(1, 3, 4))
```

ag_capture_free *Release the device memory held by a captured function*

Description

Frees every recording made by `ag_capture()`'s function `f`. The function stays usable: the next call records again. Recordings are also released when the persistent device pool is reset (device switch).

Usage

```
ag_capture_free(f)
```

Arguments

`f`	A function returned by `ag_capture`.

Value

`f`, invisibly.

ag_capture_set *Change a setting of a captured training step*

Description

Settings are inputs of the recording, read on every call, so changing one does not record again.

Usage

```
ag_capture_set(f, max_grad_norm)
```

Arguments

`f`	A function returned by `ag_capture_step`.
`max_grad_norm`	A positive number: the new gradient-clipping limit. The step must have been created with a `max_grad_norm` (clipping cannot be switched on or off after creation).

Value

`f`, invisibly.

See Also

`ag_capture_step`

ag_capture_step *Capture a whole training step as reusable device graphs*

Description

Like ag_capture, for a training step: the forward pass in fn, its backward pass and the Adam update are recorded once, and every later call is upload the inputs, two graph computes, download the outputs – no R-level op dispatch. On the CPU the wrapper runs the same step as ordinary code (zero_grad, tape, backward, step).

Usage

```
ag_capture_step(fn, params = NULL, optimizer, max_grad_norm = NULL)
```

Arguments

fn	Function of named ag_tensor arguments returning the loss (an ag_tensor), or a named list of ag_tensors with an element loss.
params	Optional list of further ag_tensors the recording references (e.g. frozen weights) and ag_buffers read with ag_get_rows; the optimizer's parameters are always included.
optimizer	An optimizer from optimizer_adam, created on the GPU when the step is to run there.
max_grad_norm	NULL (no clipping) or a positive number: clip gradients to this global norm before the update.

Details

Two graphs, not one: forward with backward, then the update. Adam writes each weight in place, and a weight is also read inside the first graph (by the forward, and by the backward of the layer below) through tensors that share its memory but are not connected to the update in the graph – so a single graph would leave the order of that read and that write to the backend.

fn is called WITHOUT with_grad_tape: the wrapper records the tape itself. Everything ag_capture requires of fn applies: data only through the arguments, no reading values back, weights on the device. The backward must be one that runs as a graph (the ops the graph backward supports).

Input names and shapes are fixed by the first call; a call with other shapes is an error (pad a short last minibatch, or run it uncaptured). The step counter and the learning rate are read from optimizer on every call, so learning-rate schedules work, and optimizer$state() stays valid. Outputs are the values of the forward pass, before the update, as in an ordinary training loop. After a captured step $grad is not set.

Supported: optimizer_adam without gradient accumulation, F32 compute dtype.

Gradient clipping by global norm (max_grad_norm): the norm of all gradients is computed in the first graph and every gradient is multiplied by min(1, max_grad_norm / (norm + 1e-6)) before the update. Branch-free, unlike clip_grad_norm: for a norm within about 1e-6 below max_grad_norm the factor is slightly under 1 instead of exactly 1. The pre-clipping norm is returned as an extra

output `grad_norm`. A non-finite norm is an error raised BEFORE the update: weights, moments and the step counter are left as they were. The moments accumulate clipped gradients, so optimizer state saved with and without clipping is not interchangeable on resume. Change the limit between calls with `ag_capture_set` (no re-recording).

Value

A function taking the same named arguments as numeric matrices and returning the loss as a matrix, or a named list of matrices.

See Also

`ag_capture`, `ag_capture_set`, `ag_capture_free`

Examples

```
W <- ag_param(matrix(rnorm(6, sd = 0.1), 2, 3))
opt <- optimizer_adam(list(W), lr = 1e-2)
train <- ag_capture_step(function(x, y) ag_mse_loss(ag_matmul(W, x), y),
                         optimizer = opt)
for (i in 1:5) loss <- train(x = matrix(1, 3, 4), y = matrix(0, 2, 4))
```

---

ag_checkpoint *Run part of the forward pass without storing its activations*

Description

Executes `fn(...)` with the gradient tape switched off, so none of the intermediate activations inside `fn` are kept. During `backward()` the segment is run again – with the same RNG state – to rebuild what the gradient rules need.

Usage

```
ag_checkpoint(fn, ...)
```

Arguments

`fn`	A function taking the tensors in `...` and returning a single `ag_tensor`. It must be reproducible: given the same inputs and RNG state it has to compute the same thing. Parameters it closes over are fine, and are given gradients as usual. One caveat, because it bites in the most natural usage: `fn` is called again during `backward()`, so it must not close over a loop variable. Wrapping layers in a `for` loop with `lyr <- layers[[i]]` in the body leaves every segment sharing one binding, and by replay time they all see the last layer – the earlier ones then get no gradient while the loss still looks perfectly normal. Capture per iteration with `local()`, as in the example below.
`...`	Input `ag_tensors`, passed to `fn`.

Details

This trades compute for memory: the segment's forward pass runs twice, and in exchange the tape never holds its activations. Use it on the deep middle of a network, not on cheap layers.

Value

An ag_tensor holding the segment's output. Its gradient flows back into both ... and any parameters used inside fn.

Examples

```
block <- ag_linear(8L, 8L, activation = "relu")
x <- ag_tensor(matrix(runif(8 * 4), 8, 4))
y <- ag_tensor(matrix(runif(8 * 4), 8, 4))
with_grad_tape({
  h    <- ag_checkpoint(function(inp) block$forward(inp), x)
  loss <- ag_mse_loss(h, y)
})
backward(loss)

# Checkpointing every second block of a deeper stack. local() gives each
# segment its own binding -- see the note on `fn` for why that matters.
layers <- lapply(1:4, function(i) ag_linear(8L, 8L, activation = "relu"))
with_grad_tape({
  h <- x
  for (i in seq_along(layers)) {
    seg <- local({
      lyr <- layers[[i]]
      function(inp) lyr$forward(inp)
    })
    h <- if (i %% 2L == 0L) ag_checkpoint(seg, h) else seg(h)
  }
  loss <- ag_mse_loss(h, y)
})
backward(loss)
```

---

ag_clamp *Element-wise clamp*

Description

Clamps values to [lo, hi]. Gradient is 1 inside the interval, 0 at the boundary (straight-through estimator).

Usage

```
ag_clamp(x, lo = -Inf, hi = Inf)
```

Arguments

x	ag_tensor
lo	Lower bound (default -Inf)
hi	Upper bound (default Inf)

Value

ag_tensor

---

ag_concat_rows *Concatenate two tensors by rows*

Description

Stacks a above b, like rbind(a, b): both must have the same number of columns (the batch). Typical use is a critic input built from observations and an action computed by the actor. The gradient of a is rows 1..nrow(a) of the incoming gradient and that of b the rest; an operand without gradient gets none.

Usage

```
ag_concat_rows(a, b)
```

Arguments

a, b	ag_tensors (or matrices) with the same number of columns

Value

ag_tensor of shape (nrow(a) + nrow(b)) x ncol(a)

See Also

ag_slice_rows

Examples

```
a <- ag_tensor(matrix(1:6, 2, 3)); b <- ag_tensor(matrix(7:9, 1, 3))
as.matrix(ag_concat_rows(a, b))
```

ag_cross_entropy_loss *Categorical Cross-Entropy loss*

Description

Generic CE: `-sum(target * log(pred)) / batch_size`. The gradient w.r.t. `pred` is `-target / pred / n`. Use `ag_softmax_cross_entropy_loss()` for the numerically stable combined softmax + CE (fused gradient `(p - y) / n`).

Usage

```
ag_cross_entropy_loss(pred, target)
```

Arguments

`pred`	ag_tensor [classes, batch_size] probabilities (any, not just softmax)
`target`	matrix [classes, batch_size] one-hot (or soft) labels

Value

scalar ag_tensor

ag_dataloader *Create a mini-batch data loader*

Description

Returns an iterator environment. Each call to `$next_batch()` returns a named list `list(x, y)` with ag_tensor objects of shape `[features, batch_size]` / `[labels, batch_size]`. After the last batch, `$has_next()` returns `FALSE`; call `$reset()` (or start a new epoch via `$epoch()`) to reshuffle and restart.

Usage

```
ag_dataloader(x, y = NULL, batch_size = 32L, shuffle = TRUE, col_major = TRUE)
```

Arguments

`x`	Feature matrix `[features, n_samples]` or `[n_samples, features]` — see `col_major`.
`y`	Label matrix with the same convention.
`batch_size`	Integer batch size.
`shuffle`	Logical; if `TRUE` (default) shuffle at each `reset()`.
`col_major`	Logical; if `TRUE` (default) `x` and `y` are already `[features, n]` (ggml/ag convention). Set `FALSE` for row-major `[n, features]` (R/Keras convention) — they will be transposed automatically.

Value

An ag_dataloader environment

Examples

```
n  <- 128L
x  <- matrix(runif(4 * n), 4, n)   # [4, 128] col-major
y  <- matrix(runif(2 * n), 2, n)
dl <- ag_dataloader(x, y, batch_size = 32L)
dl$reset()
while (dl$has_next()) {
  batch <- dl$next_batch()
  # batch$x: [4, 32],  batch$y: [2, 32]
}
```

---

ag_default_device *Return the current default compute device*

Description

Return the current default compute device

Usage

```
ag_default_device()
```

Value

"cpu" or "gpu"

---

ag_default_dtype *Return the current default dtype for GPU operations*

Description

Return the current default dtype for GPU operations

Usage

```
ag_default_dtype()
```

Value

"f32", "f16", or "bf16"

ag_device *Set the default compute device for ag_* operations*

Description

Switches all subsequent `ag_tensor` / `ag_param` operations to run on the specified device. Calling `ag_device("gpu")` initialises the best available ggml backend (Vulkan, Metal, CUDA, or CPU fallback) the first time it is called.

Usage

```
ag_device(device)
```

Arguments

`device`	`"cpu"` (default) or `"gpu"`

Details

Switching to the CPU moves every `ag_buffer` on the GPU to the host first; plan for host memory of about twice the buffers' device size (R stores the values as doubles).

Value

Invisibly the previous device string

ag_dropout *Create a Dropout layer*

Description

In training mode applies inverted dropout (random Bernoulli mask, scale by `1/(1-rate)` to preserve expected values). In eval mode is identity.

Usage

```
ag_dropout(rate)
```

Arguments

`rate`	Drop probability in [0, 1)

Value

An `ag_dropout` environment

Examples

```
drop <- ag_dropout(0.5)
x    <- ag_tensor(matrix(runif(8), 4, 2))
out  <- drop$forward(x)  # training mode by default
ag_eval(drop)
out2 <- drop$forward(x)  # identity
```

`ag_dtype` *Set the default floating-point precision for ag_* GPU operations*

Description

Controls the dtype used when uploading tensors to the ggml backend. `"bf16"` halves memory usage vs `"f32"` with minimal accuracy loss. Gradients are never computed in f16: the GPU graph backward runs in f32 only, so with `"f16"` or `"bf16"` `backward()` takes the closure path, which computes them in double – correct, but slower (the reason is counted by `ggmlR:::ag_backward_fallbacks()`). Training in f32 keeps the graph path.

Usage

```
ag_dtype(dtype)
```

Arguments

`dtype`	`"f32"` (default), `"f16"`, or `"bf16"`

Value

Invisibly the previous dtype string

`ag_embedding` *Create an Embedding layer*

Description

Maps 0-based integer indices to dense vectors via table lookup. Input: integer matrix or vector of 0-based indices. Output: float tensor `[dim, length(idx)]`.

Usage

```
ag_embedding(vocab_size, dim)
```

Arguments

`vocab_size`	Vocabulary size
`dim`	Embedding dimension

Details

Backward: scatter-add — only the looked-up rows accumulate gradient.

Value

An ag_embedding environment

Examples

```
emb <- ag_embedding(100L, 16L)
idx <- c(0L, 3L, 7L, 2L)
out <- emb$forward(idx)   # [16, 4]
```

---

ag_estimate_training_memory

Estimate the memory a training run needs

Description

Adds up the four things that are alive at the same time during training on the autograd (ag_*) path: the weights, their gradients, the optimizer's per-parameter state, and the activations held by the tape until zero_grad() clears it.

Usage

```
ag_estimate_training_memory(
  shapes,
  batch_size = 32L,
  optimizer = c("adam", "sgd", "sgd_plain"),
  activation_frac = 0.01,
  quiet = FALSE
)
```

Arguments

shapes	List of parameter shapes, each a numeric vector of dimensions (e.g. list(c(256, 128), c(128))), or a single such vector.
batch_size	Batch size the activations are sized for. Activations scale with it; weights and optimizer state do not.
optimizer	One of "adam" (keeps m and v), "sgd" (keeps a velocity, i.e. momentum > 0) or "sgd_plain" (no state).
activation_frac	Activations as a fraction of the weight bytes, per unit of batch. The default 0.01 reproduces the measured tape composition for a dense stack (inst/scripts/measure_ag_tape_memory. activations were 19% of a 4-layer 128-wide tape at batch 32). Pass your own if you have measured your model with ag_tape_memory, which is always more accurate than this scaling.

quiet — Suppress the printed report and return the figures only. Note that the double-vs-f32 caveat is part of that report, so a caller passing `quiet = TRUE` takes on reading it from the `note` field.

Details

Everything is counted at 8 bytes per scalar, because that is what the path actually uses: gradients are accumulated by R closures in double, and optimizer moments are R matrices. `ag_dtype` affects only forward tensors uploaded into a ggml buffer and does not reach either. An estimate made in f32 would be half the real figure, so the returned list carries a `note` saying so.

Value

Invisibly, a list of byte counts: `params`, `weights`, `gradients`, `optimizer`, `activations`, `total`, plus `bytes_per_scalar` and `note`. Printed as a report.

See Also

`ag_tape_memory` measures a real tape instead of estimating one; `ggml_estimate_memory` sizes a single tensor in a ggml buffer, where dtype does apply.

Examples

```
# a 3-layer MLP, 512 wide, trained with Adam at batch 64
ag_estimate_training_memory(
  shapes = list(c(512, 512), c(512, 512), c(512, 512)),
  batch_size = 64)
```

---

ag_eval *Switch a layer or sequential model to eval mode*

Description

Switch a layer or sequential model to eval mode

Usage

```
ag_eval(model)
```

Arguments

model — An ag_sequential, ag_batch_norm, or ag_dropout layer

Value

The model/layer (invisibly)

ag_exp *Element-wise exponential*

Description

Element-wise exponential

Usage

```
ag_exp(x)
```

Arguments

x	ag_tensor

Value

ag_tensor

ag_flash_attention *Multi-head attention in a single fused operation*

Description

Computes scaled dot-product attention over all heads at once with `ggml_flash_attn_ext()`, instead of the per-head loop `ag_multihead_attention` uses. Both the forward pass and its gradient are one operation each, so the tape holds a single node rather than the dozens a head loop records.

Usage

```
ag_flash_attention(q, k, v, n_heads, scale = NULL, mask = NULL, causal = FALSE)
```

Arguments

q, k, v	`ag_tensors` of shape `[d_model, seq]`. `k` and `v` may have a different sequence length from `q` (cross-attention), but must match each other.
n_heads	Number of attention heads. Must divide `d_model`.
scale	Softmax scale. Defaults to `1/sqrt(d_model / n_heads)`.
mask	Optional attention mask, `[seq_kv, seq_q]` – keys index the ROWS and queries the columns, which is the transpose of the usual "row = query" convention and is not caught by shape checking when the two lengths match. Either logical (`TRUE` where attention is allowed) or numeric (`0` to attend, `-Inf` to block). The same mask is used for the gradient.
causal	`TRUE` builds the causal mask for you: query `j` attends to keys `1..j`. Cannot be combined with `mask`.

Details

q, k and v are [d_model, seq] matrices whose rows are split into n_heads contiguous blocks – the same head layout ag_multihead_attention() uses. Projections are not included: apply W_q, W_k, W_v before the call and W_o after it.

Value

An ag_tensor of shape [d_model, seq_q].

Examples

```
d_model <- 16L; seq_len <- 8L
q <- ag_param(matrix(runif(d_model * seq_len, -1, 1), d_model, seq_len))
k <- ag_param(matrix(runif(d_model * seq_len, -1, 1), d_model, seq_len))
v <- ag_param(matrix(runif(d_model * seq_len, -1, 1), d_model, seq_len))
with_grad_tape({
  out  <- ag_flash_attention(q, k, v, n_heads = 4L)
  loss <- ag_mse_loss(out, ag_tensor(matrix(0, d_model, seq_len)))
})
backward(loss)
```

---

ag_gather *Pick one entry per column*

Description

out[1, j] = x[index[j] + 1, j]: the log-probability of the action taken in sample j, for example. Differentiable in x; the gradient of column j lands on the picked row only.

Usage

```
ag_gather(x, index)
```

Arguments

x	ag_tensor or numeric matrix, [n, batch].
index	Integer-valued vector of length ncol(x), **0-based**, each in 0..nrow(x)-1. A 1-based index is an error (out of range for n) or, worse, silently the neighbouring row – so the range is checked and NA / non-integers are rejected.

Details

Built as colSums(x * onehot(index)), so the graph backward covers it with no rule of its own. The one-hot is [n, batch] and is uploaded on every call: cheap for tens of classes (games), wasteful for thousands – there a dedicated op with an I32 index tensor is the right tool (TODO.md).

Value

ag_tensor `[1, batch]`.

Examples

```
ag_gather(matrix(1:6 + 0, 3, 2), c(2L, 0L))   # 3 4
```

`ag_get_rows` *Read buffer columns by index*

Description

Selects columns of an `ag_buffer` by 0-based index, as a device operation: on the GPU only the indices cross the bus. The name follows `ggml_get_rows`, which selects along the second ggml dimension (ne1); in the R `rows x capacity` view that is a column.

Usage

```
ag_get_rows(buf, idx)
```

Arguments

`buf`	An `ag_buffer`.
`idx`	0-based column indices: a numeric vector, or an ag_tensor with one column (f32; indices are exact below 2^{24}).

Details

The result is a `rows x length(idx)` ag_tensor without gradient, usable as an input to further ag_* operations. Inside `ag_capture()` pass `idx` as an argument of the captured function: the recording is then reused for every index vector of the same length (the batch size is part of the recording's shape), and the buffer must be listed in `params` so that freeing it invalidates the recording.

Indices given as host values are checked against the buffer and an index out of range is an error. An index tensor already on the device (inside a recording) cannot be checked; on the device every index is clamped to `0..capacity-1` before the read, so an index out of range reads the first or last column instead of memory outside the buffer.

Value

An ag_tensor of shape `rows x length(idx)`.

ag_grad *Gradient of a tensor as an R matrix*

Description

After `backward()`, `x$grad` holds the gradient – as an R matrix on the closure path, but as a device handle when the graph backward kept it on the GPU (the default there). `ag_grad()` returns the value as a matrix either way, so code that reads gradients does not depend on where they live. In graph mode it is a synchronisation point.

Usage

```
ag_grad(x)
```

Arguments

`x`	An ag_tensor (usually a parameter).

Value

A numeric matrix, or `NULL` when `x` has no gradient.

Examples

```
w <- ag_param(matrix(1:4 + 0, 2, 2))
with_grad_tape({ loss <- ag_sum(ag_mul(w, w)) })
backward(loss)
ag_grad(w)            # 2 * w
```

ag_gradcheck *Numerical gradient check (like torch.autograd.gradcheck)*

Description

Compares analytical gradients (from `backward()`) with finite-difference numerical gradients for all input tensors with `requires_grad = TRUE`.

Usage

```
ag_gradcheck(
  fn,
  inputs,
  eps = 1e-05,
  atol = 1e-04,
  verbose = FALSE,
  quiet = FALSE
)
```

Arguments

fn	A function that takes a list of ag_tensor inputs and returns a scalar ag_tensor loss (must be used inside with_grad_tape).
inputs	Named list of ag_tensor objects. Only those with requires_grad = TRUE are checked.
eps	Finite-difference step size (default 1e-5).
atol	Absolute tolerance for pass/fail (default 1e-4).
verbose	Print per-element comparison (default FALSE).
quiet	Suppress per-parameter and overall status lines (default FALSE). Useful when calling from testthat tests to keep output clean.

Value

Invisibly TRUE if all gradients match, FALSE otherwise. When quiet = FALSE (default), prints a summary report.

Examples

```
W <- ag_param(matrix(runif(6), 2, 3))
x <- ag_tensor(matrix(runif(3), 3, 1))
ag_gradcheck(
  fn = function(ins) ag_mse_loss(ag_relu(ag_matmul(ins$W, ins$x)),
                                 matrix(0, 2, 1)),
  inputs = list(W = W, x = x)
)
```

---

ag_graph_mode *Graph mode for ag_* operations on the GPU*

Description

With graph mode on, a GPU ag_* operation that has a device-resident path does not run at once: it adds its node to a queue and returns a pending value. The queue runs as ONE ggml graph when a value is read on the host (as.matrix(), a printed loss, an action index) or when backward() needs it – the graph backward then joins the same graph. Each separate launch costs on the order of a millisecond on the device, so a small network (an RL policy, a few layers) is dominated by launches, and fusing them is where the speed comes from.

Usage

```
ag_graph_mode(on, check = NULL)
```

Arguments

on	TRUE or FALSE. Missing: return the current setting.
check	TRUE/FALSE to switch the per-operation check on or off; NULL (default) leaves it as it is.

Details

Results are the same as without it. Operations that have no resident path yet still run one by one; they first run whatever is queued, so the order of computation is always the data order.

When the queue runs (synchronisation points):

- reading a value that is still queued: `as.matrix()`, printing, a loss value, `ag_argmax()`;
- an operation without a resident path (it runs the queue first);
- writing a parameter: an optimizer step, `ag_load_state_dict()`;
- the device optimizer step (it never joins the queue);
- turning graph mode off.

`backward()` does NOT force it: the graph backward appends to the queue, so forward and backward run as one graph at the next point above.

Off by default; turn it on around the code that wants it, restoring the previous value on every exit – `ag_local_mode` does that. The one thing it can break is code that orders graphs BY HAND rather than through data – reading a buffer in one graph and overwriting it in a later one with nothing linking the two. The package's own optimizer step does this and is protected; code of your own that does it must run with graph mode off. On the CPU it has no effect.

Errors. A failure inside a queued operation surfaces where the queue runs, not where the operation was called. The error names the queued operations; `check = TRUE` (or `GGMLR_AG_GRAPH_CHECK=1`) runs each operation as soon as it is queued, so the error appears at its own call – for debugging, since it gives up the fusion.

Value

The current setting when `on` is missing; otherwise the previous setting, invisibly. Turning it off runs anything still queued.

See Also

`ag_local_mode`, `ag_matmul_precision`

Examples

```
old <- ag_graph_mode(TRUE)
ag_graph_mode()
ag_graph_mode(old)
```

ag_layer_norm *Create a Layer Normalization layer*

Description

Normalizes each example independently over its `normalized_shape` features, then applies a learnable per-feature scale (`gamma`) and shift (`beta`): `y = gamma * (x - mu) / sqrt(var + eps) + beta`.

Usage

```
ag_layer_norm(normalized_shape, eps = 1e-05, elementwise_affine = TRUE)
```

Arguments

`normalized_shape`	Number of features to normalize over (the number of rows of the input).
`eps`	Small constant added to the variance for numerical stability (default 1e-5).
`elementwise_affine`	Logical; when `TRUE` (default) the layer learns `gamma` and `beta`. When `FALSE` it normalizes only and has no parameters.

Details

Unlike `ag_batch_norm`, which normalizes each feature across the batch, LayerNorm normalizes each column (example) across its features. It therefore has no running statistics and behaves identically in training and evaluation mode, which makes it the usual choice for transformers and for small or varying batch sizes.

The backward pass is exact: because `mu` and `sigma` are computed from the same example being normalized, their contribution to the gradient is included rather than treated as constant.

Value

An `ag_layer_norm` environment with `$forward()` and `$parameters()`.

See Also

`ag_batch_norm`

Examples

```
ln <- ag_layer_norm(4L)
x  <- ag_param(matrix(rnorm(8), 4, 2))   # [features, batch]
with_grad_tape({
  y    <- ln$forward(x)
  loss <- ag_mse_loss(y, matrix(0, 4, 2))
})
grads <- backward(loss)
```

`ag_linear` *Create a dense layer with learnable parameters*

Description

Returns a closure-based layer. Because ag_param uses environment semantics, the optimizer updates W and b in-place, and forward() always uses the latest weights.

Usage

```
ag_linear(
  in_features,
  out_features,
  activation = NULL,
  init = "glorot_uniform",
  gain = 1
)
```

Arguments

`in_features`	Input dimension
`out_features`	Output dimension
`activation`	"relu", "sigmoid", "tanh", "softmax", or NULL
`init`	Weight initialisation, `[out_features, in_features]`:
	`"glorot_uniform"` (default) uniform on $\pm\sqrt{6/(in+out)}$; unchanged from earlier versions, same random stream for the same seed.
	`"orthogonal"` orthogonal rows (out <= in) or columns (out > in), uniform over orthogonal matrices (QR of a Gaussian matrix with the sign of R's diagonal folded into Q). Draws from R's RNG (`rnorm`), so `set.seed` reproduces it.
	a function `function(out, in)` returning the matrix.
	a matrix used as is; no random numbers are drawn.
	The bias always starts at zero. `init` only shapes a new layer: `ag_load_model` overwrites every weight from the file.
`gain`	Multiplier for the string initialisations (e.g. `sqrt(2)` for tanh/ReLU layers, `0.01` for a policy head). A function or matrix `init` carries its own scale, so `gain` must stay 1 with them.

Value

List with `W`, `b`, `forward(x)`, `params()`

Examples

```
layer <- ag_linear(4L, 8L, activation = "relu")
x     <- ag_tensor(matrix(runif(4 * 16), 4, 16))
out   <- layer$forward(x)
head  <- ag_linear(8L, 2L, init = "orthogonal", gain = 0.01)
```

---

ag_load_model *Load an autograd module from a saved state*

Description

Reconstructs an `ag_*` module saved with `ag_save_model`. The architecture is rebuilt by calling `model_fn` (either the one passed here, or the one stored inside the container at save time), and the saved parameter and buffer values are copied back by name.

Usage

```
ag_load_model(path, model_fn = NULL, device = NULL, dtype = NULL)
```

Arguments

`path`	File path written by `ag_save_model`.
`model_fn`	Optional zero-argument rebuild function. Required if no `model_fn` was stored at save time. If both are present, this argument takes precedence.
`device`	Optional device for the rebuilt module (`"cpu"` or `"gpu"`). If `NULL` (default), the current `ag_device()` is used by the rebuild.
`dtype`	Optional GPU upload precision for the restored parameters (`"f32"`, `"f16"`, or `"bf16"`). If `NULL` (default), the dtype recorded in the file is restored, so a module saved under `ag_dtype("f16")` keeps computing in f16 regardless of the loading session's default. Pass an explicit value to override the file. Files written before dtypes were recorded fall back to whatever `model_fn` builds. Note that this is an *upload/compute* precision, not a storage precision: parameter values always live in `$data` as full-precision R numeric matrices (that is what backward needs), and `dtype` only controls the precision they are uploaded to the GPU with. Saving therefore never rounds the weights, whatever dtype is in effect.

Value

The reconstructed module with restored weights, in eval mode.

See Also

`ag_save_model`

Examples

```
build <- function() ag_sequential(ag_linear(4L, 8L), ag_linear(8L, 3L))
f <- tempfile(fileext = ".rds")
ag_save_model(build(), f, model_fn = build)
model <- ag_load_model(f)
```

`ag_load_state_dict` *Load weights into an existing model*

Description

Writes a state from `ag_state_dict` (or the container of an `ag_save_model` file read with `readRDS`) into `model` in place. Device-resident weights are overwritten in their own buffers, so an optimizer built on the model stays valid and keeps its residency. Values are converted to the model's dtype on upload.

Usage

```
ag_load_state_dict(model, state, strict = TRUE)
```

Arguments

`model`	Target model (see `ag_state_dict`).
`state`	A state dict.
`strict`	`TRUE` (default): the parameter and buffer names must match exactly. `FALSE`: names present on only one side are skipped (partial loading, transfer learning); names present on both sides are still shape-checked, always.

Details

Everything is checked before anything is written: an error leaves the model untouched.

Value

`model`, invisibly.

ag_local_mode *Set ag_* GPU modes for the rest of the calling function*

Description

Sets graph mode (ag_graph_mode) and/or matrix-multiply precision (ag_matmul_precision) and restores the previous values when the calling function exits – normally or by an error. One place for both, instead of a pair of on.exit calls per setting.

Usage

```
ag_local_mode(graph = NULL, matmul_precision = NULL, envir = parent.frame())
```

Arguments

graph	TRUE/FALSE, or NULL to leave it.
matmul_precision	"default"/"f32", or NULL to leave it.
envir	The frame whose exit restores the settings (the caller's).

Value

The previous values, invisibly.

Examples

```
f <- function() {
  ag_local_mode(graph = TRUE, matmul_precision = "f32")
  c(ag_graph_mode(), ag_matmul_precision())
}
f()                      # TRUE "f32"
ag_graph_mode()          # back to the previous value
```

ag_log *Element-wise natural logarithm*

Description

Element-wise natural logarithm

Usage

```
ag_log(x)
```

Arguments

x	ag_tensor

Value

ag_tensor

`ag_log_softmax` *Column-wise log-softmax*

Description

`log(softmax(x))` down every column, computed without forming `softmax` first: masked entries keep finite log-probabilities instead of `log(0)` = `-Inf`, so `p * logp` stays finite (entropy terms).

Usage

```
ag_log_softmax(x)
```

Arguments

`x`	ag_tensor or numeric matrix, `[n, batch]`.

Details

Mask illegal entries by ADDING a large but FINITE negative number (e.g. `-1e4`, which also fits f16). `-Inf` is not supported: the shift below multiplies x by p, and `0 * -Inf` is `NaN`. A column with every entry masked is finite but meaningless (uniform over masked actions); callers must leave at least one entry unmasked.

Stability shift. Any per-column c with `max - log(n) <= c <= max` keeps `exp(x - c)` from overflowing and the largest term from underflowing. `c = colSums(x * softmax(x))` satisfies it (`lse - c` is the entropy, at most `log(n)`, and `lse >= max`), and on the device it is three ggml ops – no max reduction, which ggml lacks along this axis. The result does not depend on c, and the gradient treats it as a constant: `dx = g - exp(logp) * colSums(g)`.

f16 compute dtype: `colSums(exp(x - c))` is at most n and is stored in the compute type, so n must stay below 65504 (the f16 maximum); larger vocabularies need `ag_dtype("f32")`. Inputs of magnitude ~1e4 are themselves quantised to a few units in f16 – a property of the input.

Value

ag_tensor of the same shape.

Examples

```
lp <- ag_log_softmax(ag_tensor(matrix(c(1, 2, 3, 0, -1e4, 0), 3, 2)))
```

ag_matmul *Matrix multiplication*

Description

Computes `A %*% B` and records the operation on the gradient tape.

Usage

```
ag_matmul(A, B)
```

Arguments

`A`	ag_tensor or numeric matrix of shape `[m, k]`
`B`	ag_tensor or numeric matrix of shape `[k, n]`

Value

ag_tensor of shape `[m, n]`

ag_matmul_precision *Matrix-multiply precision for ag_* operations on the GPU*

Description

On a Vulkan device with fp16 or cooperative-matrix support, an f32 x f32 matrix multiply with more than 8 columns runs a kernel that keeps its tiles in f16: relative error around `4e-4`, while up to 8 columns (and the CPU) give f32 accuracy (`~1e-7`). `"f32"` asks every `ag_matmul` (forward, and the input gradient of the graph backward) for the f32-tile kernel, so the result no longer depends on the batch size or the device.

Usage

```
ag_matmul_precision(precision)
```

Arguments

`precision`	`"default"` or `"f32"`. Missing: return the current setting.

Details

Cost: none measurable on small layers (RL policies, tens to hundreds of units); 2-4x slower on large ones (1024 x 1024 and up). That is why the default stays `"default"`: set it around the code that needs it and restore the previous value, e.g. `old <- ag_matmul_precision("f32"); on.exit(ag_matmul_precision(old))`. Has no effect on the CPU or with `ag_dtype("f16")` / `"bf16"`.

Value

The current setting when `precision` is missing; otherwise the previous setting, invisibly.

Examples

```
old <- ag_matmul_precision("f32")
ag_matmul_precision()
ag_matmul_precision(old)
```

`ag_max` *Column-wise maximum*

Description

`ag_gather(x, ag_argmax(x))`: the largest entry of every column, as a `[1, batch]` ag_tensor. Differentiable; the gradient goes to the selected entry only. With ties that is the FIRST maximum (the `ag_argmax` rule), and it is not split between the tied entries – a convention, which some frameworks resolve differently.

Usage

```
ag_max(x)
```

Arguments

`x`	ag_tensor or numeric matrix, `[n, batch]`.

Details

Calls `ag_argmax`, so on the GPU it is a synchronisation point.

Value

ag_tensor `[1, batch]`.

ag_mean *Mean of elements (or along a dim)*

Description

Mean of elements (or along a dim)

Usage

```
ag_mean(x, dim = NULL, keepdim = FALSE)
```

Arguments

x	ag_tensor
dim	NULL (all), 1 (row-wise), or 2 (col-wise)
keepdim	Logical

Value

ag_tensor

ag_mse_loss *Mean Squared Error loss*

Description

Mean Squared Error loss

Usage

```
ag_mse_loss(pred, target)
```

Arguments

pred	ag_tensor [units, batch_size]
target	ag_tensor or matrix [units, batch_size]

Value

scalar ag_tensor

ag_mul *Element-wise multiplication*

Description

Element-wise multiplication

Usage

```
ag_mul(A, B)
```

Arguments

A	ag_tensor or numeric matrix
B	ag_tensor or numeric matrix

Value

ag_tensor

ag_multihead_attention *Create a Multi-Head Attention layer*

Description

Implements scaled dot-product multi-head attention as in "Attention Is All You Need" (Vaswani et al., 2017).

Usage

```
ag_multihead_attention(d_model, n_heads, dropout = 0, bias = TRUE)
```

Arguments

d_model	Model (embedding) dimension
n_heads	Number of attention heads. d_model must be divisible by n_heads.
dropout	Attention dropout probability (default 0, applied in training mode only)
bias	Logical: add bias to output projection (default TRUE)

Details

Calling convention (mirrors PyTorch `nn.MultiheadAttention`):

- `layer$forward(q)` — self-attention (`k = v = q`)
- `layer$forward(q, k, v)` — cross-attention

Tensor layout: `[d_model, seq_len]` — columns are tokens, consistent with the rest of the ag_* API.

Forward pass:

```
Q = W_q %*% q            [d_k * n_heads, seq_len]
K = W_k %*% k            [d_k * n_heads, seq_len]
V = W_v %*% v            [d_v * n_heads, seq_len]

for each head h:
  q_h = Q[h*d_k+1 : (h+1)*d_k, ]    [d_k, seq_len]
  k_h = K[h*d_k+1 : (h+1)*d_k, ]    [d_k, seq_len]
  v_h = V[h*d_v+1 : (h+1)*d_v, ]    [d_v, seq_len]
  A_h = softmax(t(q_h) %*% k_h / sqrt(d_k))   [seq_len, seq_len]
  if causal_mask: A_h[i,j] = 0 for j > i
  head_h = v_h %*% A_h             [d_v, seq_len]

concat = rbind(head_1, ..., head_H)  [d_v*n_heads, seq_len]
out    = W_o %*% concat + b_o      [d_model, seq_len]
```

Value

An `ag_multihead_attention` environment with `$forward(q, k, v, causal_mask)` and `$parameters()`

Choosing between this and `ag_flash_attention()`

This layer loops over heads in R. Since `ag_*` has no slice operation, extracting one head is a matmul against a selector matrix, so a 4-head block records around 52 tape nodes for the attention core – against one for `ag_flash_attention`, which fuses all heads into a single `ggml_flash_attn_ext()` call and its gradient into one `ggml_flash_attn_back()`. Measured on the attention core alone, the fused op is 15-37x faster depending on shape (`inst/scripts/measure_ag_flash_attn.R`).

Prefer `ag_flash_attention()` for plain and causal multi-head attention. This layer remains the fallback for what the fused op cannot do:

- attention dropout (`dropout > 0`);
- packaged projections – it owns `W_q`, `W_k`, `W_v`, `W_o` and `b_o`, whereas `ag_flash_attention()` takes already-projected Q/K/V and leaves the output projection to the caller;
- shapes or options `ggml_flash_attn_back()` rejects – ALiBi, logit softcap and attention sinks all assert.

Which API this belongs to

This is an ag_* layer: it runs on the dynamic autograd path, where a tape records each operation as it executes and backward differentiates through R closures. It is not a layer for ggml_model_sequential or ggml_model and cannot be passed to them; the two APIs do not share layer objects. For attention inside a compiled model, build the block from nn_* layers instead (see the transformer encoder section of the README).

The practical consequences of being on this path: gradients and optimizer state are R doubles regardless of ag_dtype, so budget 8 bytes per scalar (ag_estimate_training_memory); the tape holds every activation until zero_grad() (ag_tape_memory); and ag_device("gpu") is usually slower than the CPU for individual ops, because each one pays a separate upload and download.

See Also

ag_flash_attention for the fused implementation.

Examples

```
# Self-attention
mha <- ag_multihead_attention(64L, 8L)
x   <- ag_tensor(matrix(rnorm(64 * 10), 64, 10))  # [d_model=64, seq_len=10]
out <- mha$forward(x)                              # [64, 10]

# Cross-attention
q   <- ag_tensor(matrix(rnorm(64 * 10), 64, 10))
kv  <- ag_tensor(matrix(rnorm(64 * 15), 64, 15))
out <- mha$forward(q, kv, kv)

# Causal (GPT-style)
out <- mha$forward(x, causal_mask = TRUE)
```

---

ag_param *Create a parameter tensor (gradient tracked)*

Description

Create a parameter tensor (gradient tracked)

Usage

```
ag_param(
  data,
  device = .ag_device_state$device,
  dtype = .ag_device_state$dtype
)
```

Arguments

data	Numeric matrix or vector
device	"cpu" (default) or "gpu"
dtype	Floating-point precision: "f32" (default), "f16", or "bf16". Ignored on CPU; controls upload precision on GPU.

Value

An ag_tensor with requires_grad = TRUE

---

ag_pow *Element-wise power*

Description

Element-wise power

Usage

```
ag_pow(x, p)
```

Arguments

x	ag_tensor
p	Numeric exponent (scalar, not tracked for gradients)

Value

ag_tensor

---

ag_relu *ReLU activation*

Description

Applies the rectified linear unit: $\max(0, x)$.

Usage

```
ag_relu(x)
```

Arguments

x	ag_tensor

Value

ag_tensor

ag_reshape *Reshape tensor*

Description

Reshape tensor

Usage

```
ag_reshape(x, nrow, ncol)
```

Arguments

x	ag_tensor
nrow	New number of rows (use -1 to infer)
ncol	New number of columns (use -1 to infer)

Value

ag_tensor with new shape, same data

ag_sample_categorical *Sample one index per column from logits*

Description

Categorical sampling by the Gumbel-max trick: `argmax(x + G)` with `G = -log(-log(u))`, `u ~ U(0, 1)`, draws index `i` of column `j` with probability `softmax(x[, j])[i]`. `x` may be logits or log-probabilities (they differ by a per-column constant).

Usage

```
ag_sample_categorical(x)
```

Arguments

x	ag_tensor or numeric matrix, `[n, batch]`.

Details

The noise comes from `stats::runif`, so `set.seed` reproduces the draw, and the CPU and GPU paths see the same noise. On the GPU only the noise goes up and only the indices come down: `x` stays on the device, and the add and argmax are one launch. Like `ag_argmax` it is a synchronisation point and not differentiable; get the log-probability of the drawn action with `ag_gather(ag_log_softmax(x), a)`.

Mask illegal entries as for `ag_log_softmax`: add a large but FINITE negative number (`-1e4`). Gumbel noise exceeds 1e4 with probability about `exp(-1e4)`, so a masked entry is never drawn while one column entry is unmasked.

Near-ties of `x + G` are rounded in f32 on the GPU and in double on the CPU, so the two paths can differ in a rare column even with the same seed.

Value

Integer vector of length `ncol(x)`, 0-based.

Examples

```
set.seed(1)
ag_sample_categorical(matrix(c(0, 0, -1e4,  5, 0, 0), 3, 2))
```

---

ag_sample_normal *Sample from a diagonal Gaussian*

Description

`mu + exp(log_std) * eps` with `eps ~ N(0, 1)` from `stats::rnorm` (so `set.seed` reproduces it). Differentiable in `mu` and `log_std` – the reparameterisation used by SAC; for PPO collection the result is just the action.

Usage

```
ag_sample_normal(mu, log_std)
```

Arguments

`mu`	ag_tensor or numeric matrix, `[n, batch]`.
`log_std`	ag_tensor or numeric matrix, `[n, batch]` or `[n, 1]`.

Details

`log_std` is either the shape of `mu` (state-dependent) or one column `[n, 1]` broadcast over the batch (state-independent). On the GPU `mu` and `log_std` stay on the device; only `eps` is uploaded.

Value

ag_tensor `[n, batch]`.

Examples

```
set.seed(1)
a <- ag_sample_normal(matrix(0, 2, 3), matrix(log(0.5), 2, 1))
```

`ag_save_model` *Save an autograd module's state to disk*

Description

Serializes the trainable parameters and persistent buffers of an `ag_sequential` (or single `ag_*` layer) module as a portable state dictionary of plain numeric matrices. This avoids serializing the live module (environments + closures), which is brittle across ggmlR versions and carries non-portable GPU pointers.

Usage

```
ag_save_model(model, path, model_fn = NULL)
```

Arguments

`model`	An `ag_sequential` module or a single `ag_*` layer exposing `parameters()`.
`path`	File path to write (an RDS container).
`model_fn`	Optional zero-argument function that rebuilds the module architecture (fresh, untrained). If supplied, it is stored in the container so `ag_load_model` can rebuild without arguments. Must not capture GPU tensors in its enclosing environment.

Details

Reconstruction requires the architecture. Either pass `model_fn` here so it is stored in the file, or pass it later to `ag_load_model`.

Value

`path`, invisibly.

See Also

`ag_load_model`

Examples

```
build <- function() ag_sequential(ag_linear(4L, 8L), ag_linear(8L, 3L))
model <- build()
ag_save_model(model, tempfile(fileext = ".rds"), model_fn = build)
```

ag_scale *Scale tensor by a scalar constant*

Description

Scale tensor by a scalar constant

Usage

```
ag_scale(x, scalar)
```

Arguments

x	ag_tensor
scalar	Numeric scalar (not tracked for gradients)

Value

ag_tensor

ag_sequential *Create a sequential container of layers*

Description

Chains layers so that `forward(x)` passes `x` through each layer in order. `parameters()` collects all trainable params from all layers. `ag_train()` / `ag_eval()` propagate mode to stateful sub-layers.

Usage

```
ag_sequential(...)
```

Arguments

...	Layer objects (ag_linear, ag_dropout, ag_batch_norm, ag_embedding) or a single list of layers.

Value

An `ag_sequential` environment

Examples

```
model <- ag_sequential(
  ag_linear(4L, 16L, activation = "relu"),
  ag_dropout(0.5),
  ag_linear(16L, 2L, activation = "softmax")
)
x   <- ag_tensor(matrix(runif(4 * 8), 4, 8))
out <- model$forward(x)
```

---

ag_sigmoid *Sigmoid activation*

Description

Applies $1/(1 + e^{-x})$.

Usage

```
ag_sigmoid(x)
```

Arguments

x	ag_tensor

Value

ag_tensor

---

ag_slice_rows *Slice a range of rows*

Description

Returns rows `from + 1` to `from + n` of `x` (`from` is 0-based), e.g. one field of a packed batch read with `ag_get_rows`. On the GPU it is a view made contiguous on the device, with nothing crossing the bus; inside `ag_capture()` `from` and `n` are constants of the recording.

Usage

```
ag_slice_rows(x, from, n)
```

Arguments

x	ag_tensor (or matrix) with at least `from + n` rows
from	0-based index of the first row
n	Number of rows

Details

The gradient is g in the sliced rows and zero elsewhere.

Value

ag_tensor of shape `n x ncol(x)`

Examples

```
x <- ag_tensor(matrix(1:12, 4, 3))
as.matrix(ag_slice_rows(x, 1, 2))
```

---

ag_softmax *Softmax activation (column-wise)*

Description

Applies numerically stable softmax along rows so that each column (one sample) sums to 1.

Usage

```
ag_softmax(x)
```

Arguments

`x`	ag_tensor of shape `[classes, batch_size]`

Value

ag_tensor of the same shape as `x`

---

ag_softmax_cross_entropy_loss *Fused softmax + cross-entropy loss (numerically stable)*

Description

Combines softmax and CE in one op using the fused gradient `(p - y) / n`. More numerically stable than chaining `ag_softmax` + `ag_cross_entropy_loss`. Use this when your last layer outputs raw logits.

Usage

```
ag_softmax_cross_entropy_loss(logits, target)
```

Arguments

logits	ag_tensor [classes, batch_size] raw (pre-softmax) scores
target	matrix [classes, batch_size] one-hot labels

Value

scalar ag_tensor

---

ag_state_dict *Model weights as plain R data*

Description

A copy of every parameter and persistent buffer (ag_batch_norm running statistics) as host matrices, keyed like model$parameters(). Values are R doubles whatever the device or ag_dtype(): an f32 value is represented exactly, so a state taken on the GPU in f32 restores on the CPU (or back) to the same numbers. f16/bf16 weights carry only their own precision; continuing exactly across f16 and f32 is not possible.

Usage

```
ag_state_dict(model)
```

Arguments

model	An ag_sequential, an ag_* layer, or anything with a parameters() or params() method.

Details

Use it for checkpoints, for copying weights between models (target networks: ag_load_state_dict(target, ag_state_dict(online))), or with saveRDS. As with any .rds, only read files you trust.

Value

A list with parameters, buffers and param_dtypes, plus format/version.

See Also

ag_load_state_dict, ag_save_model

ag_sub *Element-wise subtraction*

Description

Element-wise subtraction

Usage

```
ag_sub(A, B)
```

Arguments

A	ag_tensor or numeric matrix
B	ag_tensor or numeric matrix

Value

ag_tensor

ag_sum *Sum all elements (or along a dim): out = sum(x)*

Description

Sum all elements (or along a dim): out = sum(x)

Usage

```
ag_sum(x, dim = NULL, keepdim = FALSE)
```

Arguments

x	ag_tensor
dim	NULL (all), 1 (row-wise), or 2 (col-wise)
keepdim	Logical: keep size-1 dimensions

Value

scalar (or reduced) ag_tensor

ag_tanh *Tanh activation*

Description

Tanh activation

Usage

```
ag_tanh(x)
```

Arguments

x	ag_tensor

Value

ag_tensor

ag_tape_memory *Report what the gradient tape is holding*

Description

Walks the current tape and reports how many nodes it holds and how much memory those nodes keep alive, split into three categories: the operand tensors, which outlive the tape; the activations captured by the backward closures, which clearing the tape would release; and any extras recorded for the graph backward path beyond those.

Usage

```
ag_tape_memory(top = 10L, quiet = FALSE)
```

Arguments

top	Number of operations to list in the per-op breakdown, most expensive first. Set to 0 for totals only.
quiet	Suppress the printed report and return the figures only. For callers that consume the numbers – tests, scripts that format their own table – rather than read them.

Details

Storage shared between categories is counted once, and charged to the longest-lived one. That matters for reading the result: a weight's matrix and the `a_snap` an `ag_matmul` closure captured are the same object, so it is reported as an operand. The activation figure is therefore what clearing the tape would actually free, not the sum of everything the closures reference.

The tape lives from the start of `with_grad_tape` until `zero_grad()` clears it, which happens after the optimizer step. Call this between `backward` and `zero_grad()` to see the peak.

Value

Invisibly, a list with nodes, bytes_total, bytes_snapshots, bytes_fields, bytes_inputs and a data frame by_op. Printed as a report as a side effect.

Examples

```
w <- ag_param(matrix(runif(64), 8, 8))
x <- ag_tensor(matrix(runif(64), 8, 8))
with_grad_tape({
  h    <- ag_relu(ag_matmul(w, x))
  loss <- ag_mse_loss(h, matrix(0, 8, 8))
})
backward(loss)
ag_tape_memory()
```

---

ag_tensor *Create a dynamic tensor (no gradient tracking)*

Description

ag_tensor is backed by an R environment so all references to the same tensor see updates (like PyTorch tensors).

Usage

```
ag_tensor(
  data,
  device = .ag_device_state$device,
  dtype = .ag_device_state$dtype
)
```

Arguments

data	Numeric matrix or vector
device	"cpu" (default) or "gpu". When "gpu", compute operations will be dispatched to the ggml backend.
dtype	Floating-point precision: "f32" (default), "f16", or "bf16". Ignored on CPU; controls upload precision on GPU.

Value

An ag_tensor object (environment)

ag_to_device *Move a tensor to the specified device*

Description

Copies an ag_tensor to the target device, returning a new tensor. The original tensor is not modified.

Usage

```
ag_to_device(tensor, device)
```

Arguments

tensor	An ag_tensor
device	"cpu" or "gpu"

Value

A new ag_tensor on the target device (or the original if already on the target device)

ag_train *Switch a layer or sequential model to training mode*

Description

Switch a layer or sequential model to training mode

Usage

```
ag_train(model)
```

Arguments

model	An ag_sequential, ag_batch_norm, or ag_dropout layer

Value

The model/layer (invisibly)

ag_transpose *Transpose a tensor*

Description

Transpose a tensor

Usage

```
ag_transpose(x)
```

Arguments

x	ag_tensor

Value

ag_tensor with rows and columns swapped

as.matrix.ag_tensor *Extract the value of an ag_tensor as an R matrix*

Description

The value of an ag_tensor lives in $data only on the CPU. On the GPU it lives in a backend buffer and $data stays NULL until something materialises it, so reading $data directly works on one device and returns NULL on the other. This method downloads the value when it has to and returns the same matrix either way – use it rather than $data in code that may run on a GPU.

Usage

```
## S3 method for class 'ag_tensor'
as.matrix(x, ...)
```

Arguments

x	An ag_tensor
...	Ignored

Value

A numeric matrix holding the tensor's current value.

Examples

```
t <- ag_tensor(matrix(1:6, nrow = 2))
as.matrix(t)
```

as.matrix.ggml_matrix *Extract the underlying matrix from a ggml_matrix*

Description

Extract the underlying matrix from a ggml_matrix

Usage

```
## S3 method for class 'ggml_matrix'
as.matrix(x, ...)
```

Arguments

x	A ggml_matrix.
...	Ignored; present for S3 as.matrix compatibility.

Value

The plain numeric matrix.

as_gpu_matrix *Wrap a matrix so multiplies run on the GPU*

Description

Tags a matrix as GPU-backed. In an expression like as_gpu_matrix(A) %*% B, the multiply dispatches to ggml_matmul (Vulkan GPU with CPU fallback) instead of R's built-in. Only one operand needs wrapping. crossprod() and tcrossprod() on a wrapped matrix likewise use the GPU.

Usage

```
as_gpu_matrix(x, device = "auto", prec = "f32")
```

Arguments

x	A numeric matrix (or something coercible with as.matrix).
device	"auto" (default), "gpu" or "cpu"; see ggml_matmul.
prec	"f32" (default) or "f16"; see ggml_matmul.

Value

A ggml_matrix.

Examples

```
A <- matrix(rnorm(9), 3)
B <- matrix(rnorm(9), 3)
as_gpu_matrix(A, device = "cpu") %*% B
```

`augment.ggmlr_parsnip_model` *Augment new data with predictions from a fitted ggml parsnip model*

Description

Adds prediction columns to ‘new_data‘, broom style. For classification this appends ‘.pred_class‘ plus one ‘.pred_<level>‘ probability column per class; for regression it appends ‘.pred‘. Predictions are produced by the existing ‘predict()‘ method for ggml parsnip models (no duplicate inference logic).

Usage

```
## S3 method for class 'ggmlr_parsnip_model'
augment(x, new_data, ...)
```

Arguments

`x`	A fitted ‘ggmlr_parsnip_model‘ (the engine object inside a parsnip fit; e.g. from ‘extract_fit_engine()‘).
`new_data`	A data frame of predictors (same columns used for fitting).
`...`	Unused; for generic compatibility.

Value

‘new_data‘ as a tibble with prediction columns appended.

Examples

```
ggml_set_n_threads(1L)  # deterministic, single OpenMP pool
spec <- parsnip::mlp(hidden_units = 8L, epochs = 3L) |>
  parsnip::set_engine("ggml", backend = "cpu") |>
  parsnip::set_mode("regression")
fit_obj <- parsnip::fit(spec, mpg ~ ., data = mtcars)
generics::augment(parsnip::extract_fit_engine(fit_obj), mtcars)
```

backward *Run backward pass from a scalar loss tensor*

Description

Traverses the gradient tape in reverse and accumulates gradients into `tensor$grad` for all leaf tensors with `requires_grad = TRUE`.

Usage

```
backward(loss)
```

Arguments

`loss`	Scalar ag_tensor

Value

Named environment: tensor id -> gradient matrix (for use by optimizer$step)

Examples

```
w <- ag_param(matrix(runif(4), 2, 2))
x <- ag_tensor(matrix(c(1, 2), 2, 1))
y <- ag_tensor(matrix(c(0, 1), 2, 1))
with_grad_tape({
  out  <- ag_matmul(w, x)
  loss <- ag_mse_loss(out, y)
})
grads <- backward(loss)
```

check_grad_anomaly *Detect NaN / Inf gradients*

Description

Scans the gradient environment produced by `backward()` and reports every parameter whose gradient contains `NaN` or `Inf` values. Useful as a debugging hook right after `backward()` — it pinpoints the offending layer instead of leaving you with a model that silently turns to `NaN` several epochs later.

Usage

```
check_grad_anomaly(
  params,
  grads,
  action = c("warn", "stop", "silent"),
  max_abs = NULL
)
```

Arguments

params	List of ag_param tensors (same as passed to optimizer), named or not.
grads	Gradient environment returned by backward().
action	One of "warn" (default), "stop" or "silent" — what to do when an anomaly is found.
max_abs	Optional numeric threshold; gradients whose maximum absolute finite value exceeds it are also reported (as "large"). NULL (default) disables this check.

Value

A data frame (invisibly) with one row per parameter that has a gradient, containing columns param, n_nan, n_inf, max_abs and status. status is one of "ok", "nan", "inf", "nan+inf" or "large". Parameters without a gradient are reported with status = "missing".

Examples

```
w  <- ag_param(matrix(runif(4), 2, 2))
x  <- ag_tensor(matrix(c(1, 1), 2, 1))
with_grad_tape({
  out  <- ag_matmul(w, x)
  loss <- ag_mse_loss(out, matrix(0, 2, 1))
})
grads <- backward(loss)
check_grad_anomaly(list(w = w), grads)
```

---

clip_grad_norm *Clip gradients by global L2 norm*

Description

Rescales all gradients in grads so that their global L2 norm does not exceed max_norm. Modifies the grads environment in-place and returns the pre-clip norm.

Usage

```
clip_grad_norm(params, grads, max_norm)
```

Arguments

params	List of ag_param tensors (same as passed to optimizer), named or not.
grads	Gradient environment returned by backward().
max_norm	Maximum allowed global L2 norm.

Details

Call this **after** backward() and **before** optimizer$step().

Value

Numeric: the global L2 norm before clipping (invisibly).

Examples

```
w  <- ag_param(matrix(runif(4), 2, 2))
x  <- ag_tensor(matrix(c(1, 1), 2, 1))
with_grad_tape({
  out  <- ag_matmul(w, x)
  loss <- ag_mse_loss(out, matrix(0, 2, 1))
})
grads <- backward(loss)
clip_grad_norm(list(w = w), grads, max_norm = 1.0)
```

---

clip_grad_value *Clip gradients element-wise by value*

Description

Clamps every gradient element into [-clip_value, clip_value]. Unlike clip_grad_norm this does not preserve the gradient direction — each element is treated independently. Modifies the grads environment in-place.

Usage

```
clip_grad_value(params, grads, clip_value)
```

Arguments

params	List of ag_param tensors (same as passed to optimizer), named or not.
grads	Gradient environment returned by backward().
clip_value	Positive numeric; maximum absolute value per element.

Details

Call this **after** backward() and **before** optimizer$step().

Value

Numeric: the largest absolute gradient value before clipping (invisibly). Returns 0 when no gradients were found.

See Also

clip_grad_norm

Examples

```
w  <- ag_param(matrix(runif(4), 2, 2))
x  <- ag_tensor(matrix(c(1, 1), 2, 1))
with_grad_tape({
  out  <- ag_matmul(w, x)
  loss <- ag_mse_loss(out, matrix(0, 2, 1))
})
grads <- backward(loss)
clip_grad_value(list(w = w), grads, clip_value = 0.5)
```

---

compile.ggml_sequential_model

Compile a Model

Description

Configures the model for training by setting the optimizer, loss function, and metrics. This is the keras-compatible interface; it delegates to ggml_compile.

Usage

```
## S3 method for class 'ggml_sequential_model'
compile(
  object,
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"),
  ...
)

## S3 method for class 'ggml_functional_model'
compile(
  object,
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"),
  ...
)
```

Arguments

object	A model object (e.g. ggml_sequential_model or ggml_functional_model).
optimizer	Character: "adam", "adamw", or "sgd".
loss	Character: "categorical_crossentropy", "mse", "mae", "huber" or "binary_crossentropy". For a multi-output model, a list with one loss per head.
metrics	Character vector of metrics (default "accuracy"); see ggml_compile for the full list.
...	Additional arguments passed to ggml_compile.

Value

The compiled model (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(10, activation = "softmax", input_shape = 4)
model <- compile(model, optimizer = "adam",
                 loss = "categorical_crossentropy")
```

---

dequantize_row_iq2_xxs

Dequantize Row (IQ)

Description

Converts IQ (integer quantization) data back to float values. IQ formats provide high compression with importance-matrix-aware quantization.

Usage

```
dequantize_row_iq2_xxs(raw_data, n_elements)

dequantize_row_iq2_xs(raw_data, n_elements)

dequantize_row_iq2_s(raw_data, n_elements)

dequantize_row_iq3_xxs(raw_data, n_elements)

dequantize_row_iq3_s(raw_data, n_elements)

dequantize_row_iq4_nl(raw_data, n_elements)

dequantize_row_iq4_xs(raw_data, n_elements)
```

```
dequantize_row_iq1_s(raw_data, n_elements)

dequantize_row_iq1_m(raw_data, n_elements)
```

Arguments

raw_data	Raw vector containing quantized data
n_elements	Number of elements to dequantize

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

dequantize_row_mxfp4 *Dequantize Row (MXFP4)*

Description

Converts MXFP4 (microscaling FP4) quantized data back to float values.

Usage

```
dequantize_row_mxfp4(raw_data, n_elements)
```

Arguments

raw_data	Raw vector containing quantized data
n_elements	Number of elements to dequantize

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

dequantize_row_nvfp4 *Dequantize NVFP4 Data*

Description

Converts NVFP4 quantized data back to float values.

Usage

```
dequantize_row_nvfp4(raw_data, n_elements)
```

Arguments

raw_data	Raw vector of NVFP4 quantized data
n_elements	Number of dequantized elements (must be multiple of 64)

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

dequantize_row_q1_0 *Dequantize Q1_0 Data*

Description

Converts Q1_0 quantized data back to float values.

Usage

```
dequantize_row_q1_0(raw_data, n_elements)
```

Arguments

raw_data	Raw vector of Q1_0 quantized data
n_elements	Number of dequantized elements (must be multiple of 128)

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

dequantize_row_q2_K *Dequantize Row (K-quants)*

Description

Converts K-quant quantized data back to float values. K-quants (q2_K through q8_K) provide better quality/size tradeoffs.

Usage

```
dequantize_row_q2_K(raw_data, n_elements)

dequantize_row_q3_K(raw_data, n_elements)

dequantize_row_q4_K(raw_data, n_elements)
```

```
dequantize_row_q5_K(raw_data, n_elements)

dequantize_row_q6_K(raw_data, n_elements)

dequantize_row_q8_K(raw_data, n_elements)
```

Arguments

raw_data	Raw vector containing quantized data
n_elements	Number of elements to dequantize

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

dequantize_row_q4_0 *Dequantize Row (Q4_0)*

Description

Converts Q4_0 quantized data back to float values.

Usage

```
dequantize_row_q4_0(raw_data, n_elements)

dequantize_row_q4_1(raw_data, n_elements)

dequantize_row_q5_0(raw_data, n_elements)

dequantize_row_q5_1(raw_data, n_elements)

dequantize_row_q8_0(raw_data, n_elements)
```

Arguments

raw_data	Raw vector containing quantized data
n_elements	Number of elements to dequantize

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

dequantize_row_tq1_0 *Dequantize Row (Ternary)*

Description

Converts ternary quantized data back to float values. TQ1_0 and TQ2_0 are extreme compression formats.

Usage

```
dequantize_row_tq1_0(raw_data, n_elements)

dequantize_row_tq2_0(raw_data, n_elements)
```

Arguments

raw_data	Raw vector containing quantized data
n_elements	Number of elements to dequantize

Value

Numeric vector of dequantized values

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

dp_train *Data-parallel training across multiple GPUs*

Description

Runs synchronous data-parallel training:

1. `make_model()` is called `n_gpu` times to create one independent model replica per GPU (each with its own parameters).
2. Each iteration: the current data item is forwarded through every replica in parallel; gradients are computed via `backward()`.
3. Gradients are averaged across all replicas (element-wise mean).
4. One optimizer step is taken on replica 0; updated weights are then broadcast to replicas 1 … N-1 so all replicas stay in sync.

Usage

```
dp_train(
  make_model,
  data,
  loss_fn = NULL,
  forward_fn = NULL,
  target_fn = NULL,
  n_gpu = NULL,
  n_iter = 10L,
  lr = 0.001,
  max_norm = Inf,
  verbose = 10L
)
```

Arguments

`make_model`	A zero-argument function that returns a model object with at least `$forward(x)` and `$parameters()` methods. Called `n_gpu` times; each call must produce independent parameters.
`data`	A list of training samples. Each element is passed directly to `forward_fn` (or to `model$forward()` if `forward_fn` is `NULL`).
`loss_fn`	A function `(logits, target) -> scalar ag_tensor`. If `NULL`, `forward_fn` must return the loss directly.
`forward_fn`	Optional function `(model, sample) -> logits`. If `NULL`, the sample is passed directly as `model$forward(sample)`.
`target_fn`	Optional function `(sample) -> target`. Used when `loss_fn` is not `NULL` to extract the target from a sample. If `NULL`, `sample` itself is used as the target.
`n_gpu`	Number of GPU replicas (default: all available Vulkan devices, minimum 1).
`n_iter`	Number of training iterations (passes over `data`).

lr	Learning rate for Adam optimizer (default 1e-3).
max_norm	Gradient clipping threshold (default Inf = no clip).
verbose	Print loss every verbose iterations, or FALSE to suppress output.

Details

Because all replicas live in the same R process and ag_param uses environment (reference) semantics, no IPC or NCCL is required — weight synchronisation is a simple in-place copy.

Value

A list with:

params Named list of final parameters (from replica 0).

loss_history Numeric vector of per-iteration mean loss.

model Replica 0 model object.

Examples

```
make_model <- function() {
  W <- ag_param(matrix(rnorm(4), 2, 2))
  list(
    forward    = function(x) ag_matmul(W, x),
    parameters = function() list(W = W)
  )
}
data <- lapply(1:8, function(i) matrix(rnorm(2), 2, 1))
result <- dp_train(
  make_model = make_model,
  data       = data,
  loss_fn    = function(out, tgt) ag_mse_loss(out, tgt),
  target_fn  = function(s) s,
  n_gpu      = 1L,
  n_iter     = 10L,
  lr         = 1e-3,
  verbose    = FALSE
)
```

---

evaluate *Evaluate a Model*

Description

Generic for computing loss and metrics on test data.

Usage

```
evaluate(object, ...)

## S3 method for class 'ggml_sequential_model'
evaluate(object, x, y, batch_size = NULL, ...)

## S3 method for class 'ggml_functional_model'
evaluate(object, x, y, batch_size = NULL, ...)
```

Arguments

object	A trained model object.
...	Additional arguments passed to ggml_evaluate.
x	Test data.
y	Test labels.
batch_size	Batch size. NULL (default) uses 32, as in keras3.

Value

A named list with loss and metric values.

Relationship to the generics package

generics::evaluate declares its first argument as x, so S3 dispatch happens on whatever is bound to x. With the keras3 argument order evaluate(model, x, y) that works positionally, but naming the data explicitly – evaluate(model, x = x, y = y) – binds the *data* to x, dispatches on it, and fails with "no applicable method". Declaring the generic here with object as its first argument fixes that and also resolves the S3 generic/method consistency warning from R CMD check.

The trade-off: attaching both ggmlR and keras3 makes the two evaluate generics mask one another, and the later-attached package wins. Qualify the call (ggmlR::evaluate(...)) if both are loaded. compile() and fit() are unaffected – generics declares those with object first, so they are still re-exported from there.

---

fit.ggml_sequential_model *Train a Model*

Description

Trains the model on data for a fixed number of epochs. This is the keras-compatible interface; it delegates to ggml_fit.

Usage

```
## S3 method for class 'ggml_sequential_model'
fit(
  object,
  x,
  y,
  epochs = 1L,
  batch_size = 32L,
  validation_split = 0,
  validation_data = NULL,
  verbose = 1L,
  shuffle = TRUE,
  callbacks = list(),
  ...
)

## S3 method for class 'ggml_functional_model'
fit(
  object,
  x,
  y,
  epochs = 1L,
  batch_size = 32L,
  validation_split = 0,
  validation_data = NULL,
  verbose = 1L,
  shuffle = TRUE,
  callbacks = list(),
  ...
)
```

Arguments

`object`	A compiled model object.
`x`	Training data. Matrix, array, or list of matrices (multi-input).
`y`	Training labels (matrix, one-hot encoded for classification).
`epochs`	Number of training epochs (default 1).
`batch_size`	Batch size (default 32).
`validation_split`	Fraction of data for validation (default 0).
`validation_data`	Optional `list(x_val, y_val)`.
`verbose`	0 = silent, 1 = progress (default 1).
`shuffle`	Shuffle the training data (default `TRUE`, as in keras3). Set to `FALSE` for time series or exactly reproducible runs.
`callbacks`	List of callback objects (default `list()`).
`...`	Additional arguments passed to `ggml_fit`.

Value

The trained model (invisibly), with `model$history`.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(10, activation = "softmax", input_shape = 4)
model <- compile(model, optimizer = "adam",
                 loss = "categorical_crossentropy")
# model <- fit(model, x_train, y_train, epochs = 5, batch_size = 32)
```

---

ggml_abort_is_r_enabled *Check if R Abort Handler is Enabled*

Description

Check if R Abort Handler is Enabled

Usage

```
ggml_abort_is_r_enabled()
```

Value

Logical indicating if R-compatible abort handling is active

See Also

Other logging: `ggml_log_is_r_enabled()`, `ggml_log_set_default()`, `ggml_log_set_r()`, `ggml_set_abort_callback`
`ggml_set_abort_callback_r()`

---

ggml_abs *Absolute Value (Graph)*

Description

Creates a graph node for element-wise absolute value: |x|

Usage

```
ggml_abs(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the abs operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(-2, -1, 1, 2))
result <- ggml_abs(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [2, 1, 1, 2]
ggml_free(ctx)
```

---

ggml_abs_inplace *Absolute Value In-place (Graph)*

Description

Creates a graph node for in-place element-wise absolute value.

Usage

```
ggml_abs_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with absolute values

ggml_add *Add tensors*

Description

Creates a graph node for element-wise addition. Must be computed using ggml_build_forward_expand() and ggml_graph_compute().

Usage

```
ggml_add(ctx, a, b)

ggml_add(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor
b	Second tensor (same shape as a)

Value

Tensor representing the addition operation

Tensor representing the addition operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(5, 4, 3, 2, 1))
result <- ggml_add(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)


ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(5, 4, 3, 2, 1))
result <- ggml_add(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
```

```
ggml_free(ctx)
```

---

ggml_add_inplace *Element-wise Addition In-place (Graph)*

Description

Creates a graph node for in-place element-wise addition. Result is stored in tensor a, saving memory allocation. Returns a view of the modified tensor.

Usage

```
ggml_add_inplace(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor (will be modified in-place)
b	Second tensor (same shape as a)

Value

View of tensor a with the addition result

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(5, 4, 3, 2, 1))
result <- ggml_add_inplace(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

ggml_add_rel_pos *Add Relative Position Bias (Graph)*

Description

Adds width and height relative-position bias to a.

Usage

```
ggml_add_rel_pos(ctx, a, pw, ph)
```

Arguments

ctx	GGML context
a	Input tensor
pw	Width relative-position tensor
ph	Height relative-position tensor

Value

Tensor with added relative-position bias

ggml_add1 *Add Scalar to Tensor (Graph)*

Description

Creates a graph node for adding a scalar (1-element tensor) to all elements of a tensor. This is more efficient than creating a full tensor of the same value.

Usage

```
ggml_add1(ctx, a, b)
```

Arguments

ctx	GGML context
a	Input tensor
b	Scalar tensor (1-element tensor)

Value

Tensor representing the operation a + b (broadcasted)

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
scalar <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 1)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(scalar, 10)
result <- ggml_add1(ctx, a, scalar)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

---

ggml_apply *Apply a Layer Object to a Tensor Node*

Description

Applies a `ggml_layer` object (created with `ggml_dense()`, `ggml_lstm()`, etc.) to a `ggml_tensor_node`. Applying the *same* layer object to multiple tensor nodes produces shared weights – the identity of the layer object (`layer$layer_id`) is used as the sharing key, not its name.

Usage

```
ggml_apply(tensor, layer)
```

Arguments

`tensor`	A `ggml_tensor_node` (e.g. from `ggml_input()`).
`layer`	A `ggml_layer` object.

Value

A new `ggml_tensor_node`.

Examples

```
encoder <- ggml_dense(64L, activation = "relu")
x1 <- ggml_input(shape = 32L)
x2 <- ggml_input(shape = 32L)
out1 <- x1 |> ggml_apply(encoder)
out2 <- x2 |> ggml_apply(encoder)  # shared weights
model <- ggml_model(inputs = list(x1, x2),
                    outputs = list(out1, out2))
```

ggml_arange *Arange (Graph)*

Description

Creates a 1D F32 tensor with values from `start` (inclusive) to `stop` (exclusive) in steps of `step`.

Usage

```
ggml_arange(ctx, start, stop, step = 1)
```

Arguments

`ctx`	GGML context
`start`	Start value (inclusive)
`stop`	Stop value (exclusive)
`step`	Step between values (default 1)

Value

1D F32 tensor

ggml_are_same_layout *Check if Two Tensors Have the Same Layout*

Description

Compares two tensors to check if they have identical type, shape, and strides. Tensors with the same layout can be used interchangeably for memory operations.

Usage

```
ggml_are_same_layout(a, b)
```

Arguments

`a`	External pointer to first tensor
`b`	External pointer to second tensor

Value

Logical indicating if tensors have identical layout

See Also

Other tensor: ggml_get_op_params(), ggml_get_op_params_f32(), ggml_get_op_params_i32(), ggml_set_op_params(), ggml_set_op_params_f32(), ggml_set_op_params_i32()

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 4)
b <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 4)
same <- ggml_are_same_layout(a, b)  # TRUE
ggml_free(ctx)
```

---

ggml_are_same_shape *Compare Tensor Shapes*

Description

Checks if two tensors have the same shape.

Usage

```
ggml_are_same_shape(a, b)
```

Arguments

a	First tensor
b	Second tensor

Value

TRUE if shapes are identical, FALSE otherwise

---

ggml_are_same_stride *Compare Tensor Strides*

Description

Check if two tensors have the same stride pattern. Useful for determining if tensors can share operations.

Usage

```
ggml_are_same_stride(a, b)
```

Arguments

a	First tensor
b	Second tensor

Value

Logical indicating if strides are identical

See Also

Other tensor_layout: ggml_can_repeat(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_2(), ggml_is_contiguous_channels(), ggml_is_contiguous_rows(), ggml_is_contiguously_allocated()

---

ggml_argmax *Argmax (Graph)*

Description

Creates a graph node that finds the index of the maximum value. CRITICAL for token generation in LLMs.

Usage

```
ggml_argmax(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor with argmax indices

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 5, 3, 2, 4))
result <- ggml_argmax(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_i32(result)  # 1 (0-indexed)
ggml_free(ctx)
```

ggml_argsort *Argsort - Get Sorting Indices (Graph)*

Description

Returns indices that would sort the tensor rows. Each row is sorted independently.

Usage

```
ggml_argsort(ctx, a, order = GGML_SORT_ORDER_ASC)
```

Arguments

ctx	GGML context
a	Input tensor to sort (F32)
order	Sort order: GGML_SORT_ORDER_ASC (0) or GGML_SORT_ORDER_DESC (1)

Value

Tensor of I32 indices that would sort each row

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Create tensor with values to sort
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(3, 1, 4, 1, 5))
# Get indices for ascending sort
indices <- ggml_argsort(ctx, a, GGML_SORT_ORDER_ASC)
graph <- ggml_build_forward_expand(ctx, indices)
ggml_graph_compute(ctx, graph)
result <- ggml_get_i32(indices)
# result: [1, 3, 0, 2, 4] (0-indexed positions for sorted order)
ggml_free(ctx)
```

ggml_attention *Create a Multi-Head Attention Layer Object*

Description

Returns a reusable layer object for use with ggml_apply, the scaled dot-product attention of *Attention Is All You Need* with n_heads heads computed in parallel. Applying the same object to several tensor nodes shares one set of projection weights, which is how an encoder block is reused across a stack.

Usage

```
ggml_attention(
  d_model,
  n_heads = 1L,
  causal = FALSE,
  bias = TRUE,
  rope = FALSE,
  rope_base = 10000,
  dropout = 0,
  name = NULL,
  trainable = TRUE
)
```

Arguments

d_model	Integer, model width. Must be divisible by n_heads and must match the parent's feature dimension.
n_heads	Integer, number of attention heads (default 1).
causal	Logical: mask out positions after the current one, so a query attends only to keys at or before it (default FALSE). This is what makes a decoder autoregressive. Only meaningful for self-attention – masking a cross-attention score matrix by position compares indices in two unrelated sequences, so it is rejected rather than silently applied.
bias	Logical: add a bias to the output projection (default TRUE).
rope	Logical: apply rotary position embedding to queries and keys (default FALSE). Weightless and relative, so it generalises to unseen sequence lengths. Needs an even d_model / n_heads.
rope_base	Rotation base for rope (default 10000).
dropout	Dropout rate applied to the attention weights during training (default 0, no dropout).
name	Optional character name.
trainable	Logical; whether weights are updated during training.

Details

For the pipe-style form that appends attention to a graph (x |> ggml_layer_attention(...)), see ggml_layer_attention.

Value

A ggml_layer object.

Shapes

The parent must be a sequence node of shape c(seq_len, d_model) – the same layout ggml_layer_lstm takes – and the output has that same shape, so attention blocks stack directly.

Self- and cross-attention

Applied to a single node the layer is self-attention: queries, keys and values all come from that node. Applied to a list of two nodes, `ggml_apply(list(query, context), attn)`, it is cross-attention: queries come from the first, keys and values from the second, which may have a different sequence length (but must share `d_model`). This is the decoder-to-encoder path of a full transformer.

Weights

Four projections `W_q`, `W_k`, `W_v`, `W_o`, each `d_model x d_model`, plus an output bias `b_o` when `bias = TRUE`. Glorot-uniform initialised, zero bias.

See Also

`ggml_layer_attention`, `ggml_apply`

Examples

```
# One attention block shared between two sequences.
attn <- ggml_attention(d_model = 32L, n_heads = 4L)
x1   <- ggml_input(shape = c(10L, 32L))
x2   <- ggml_input(shape = c(10L, 32L))
o1   <- ggml_apply(x1, attn)
o2   <- ggml_apply(x2, attn)          # shared weights
stopifnot(identical(o1$layer_id, o2$layer_id))

# Cross-attention: queries from x1, keys/values from a longer context.
ctx  <- ggml_input(shape = c(15L, 32L))
o3   <- ggml_apply(list(x1, ctx), ggml_attention(32L, 4L))
```

---

ggml_backend_alloc_ctx_tensors

Allocate Context Tensors to Backend

Description

Allocates all tensors in a GGML context to a specific backend. Returns a buffer that must be freed when no longer needed.

Usage

```
ggml_backend_alloc_ctx_tensors(ctx, backend)
```

Arguments

`ctx`	GGML context
`backend`	Backend handle

Value

Backend buffer object

`ggml_backend_buffer_clear` *Clear buffer memory*

Description

Clear buffer memory

Usage

```
ggml_backend_buffer_clear(buffer, value = 0L)
```

Arguments

`buffer`	External pointer to buffer
`value`	Byte value to fill with (default 0)

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host(), ggml_backend_buffer_is_mu
ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(),
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_all
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_buffer_free *Free Backend Buffer*

Description

Frees a backend buffer and all associated memory.

Usage

```
ggml_backend_buffer_free(buffer)
```

Arguments

buffer	Backend buffer object

Value

No return value, called for side effects

ggml_backend_buffer_get_size *Get Backend Buffer Size*

Description

Returns the total size of a backend buffer.

Usage

```
ggml_backend_buffer_get_size(buffer)
```

Arguments

buffer	Backend buffer object

Value

Size in bytes

ggml_backend_buffer_get_usage *Get buffer usage*

Description

Get buffer usage

Usage

```
ggml_backend_buffer_get_usage(buffer)
```

Arguments

buffer	External pointer to buffer

Value

Usage constant

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_is_host(), ggml_backend_buffer_is_multi_
ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(),
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_all
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_buffer_is_host` *Check if buffer is host memory*

Description

Check if buffer is host memory

Usage

```
ggml_backend_buffer_is_host(buffer)
```

Arguments

`buffer`	External pointer to buffer

Value

Logical indicating if buffer is in host memory

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_mult
ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(),
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_all
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_buffer_is_multi_buffer` *Check if buffer is a multi-buffer*

Description

Check if buffer is a multi-buffer

Usage

```
ggml_backend_buffer_is_multi_buffer(buffer)
```

Arguments

`buffer`	External pointer to buffer

Value

Logical indicating if buffer is a multi-buffer

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(),
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_all
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_buffer_name *Get Backend Buffer Name*

Description

Returns the name/type of a backend buffer.

Usage

```
ggml_backend_buffer_name(buffer)
```

Arguments

buffer	Backend buffer object

Value

Character string with buffer name

ggml_backend_buffer_reset *Reset buffer*

Description

Reset buffer

Usage

```
ggml_backend_buffer_reset(buffer)
```

Arguments

buffer	External pointer to buffer

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_buffer_set_usage

Set buffer usage hint

Description

Set buffer usage hint

Usage

```
ggml_backend_buffer_set_usage(buffer, usage)
```

Arguments

buffer	External pointer to buffer
usage	Usage constant (use ggml_backend_buffer_usage_* functions)

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_usage_any(),
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_all
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_buffer_usage_any

Buffer usage: Any

Description

Buffer usage: Any

Usage

```
ggml_backend_buffer_usage_any()
```

Value

Integer constant for any buffer usage

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignm
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),

ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(), ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_buffer_usage_compute` *Buffer usage: Compute*

Description

Buffer usage: Compute

Usage

```
ggml_backend_buffer_usage_compute()
```

Value

Integer constant for compute buffer usage

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_weights(), ggml_backend_buft_get_alignment
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(), ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),

ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_buffer_usage_weights
Buffer usage: Weights

Description

Buffer usage: Weights

Usage

```
ggml_backend_buffer_usage_weights()
```

Value

Integer constant for weights buffer usage

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buft_get_alignment
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_buft_get_alignment *Get the allocation alignment of a buffer type*

Description

Get the allocation alignment of a buffer type

Usage

```
ggml_backend_buft_get_alignment(buft)
```

Arguments

buft	External pointer to buffer type

Value

Numeric alignment in bytes

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_buft_get_max_size` *Get the maximum allocation size of a buffer type*

Description

Get the maximum allocation size of a buffer type

Usage

```
ggml_backend_buft_get_max_size(buft)
```

Arguments

`buft`	External pointer to buffer type

Value

Numeric maximum single-allocation size in bytes

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_is_host(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_buft_is_host` *Check whether a buffer type allocates host-accessible memory*

Description

Check whether a buffer type allocates host-accessible memory

Usage

```
ggml_backend_buft_is_host(buft)
```

Arguments

`buft`	External pointer to buffer type

Value

Logical, TRUE if buffers of this type live in host memory

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_name(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_buft_name *Get the name of a buffer type*

Description

Get the name of a buffer type

Usage

```
ggml_backend_buft_name(buft)
```

Arguments

buft	External pointer to buffer type

Value

Character name of the buffer type

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_cpu_init *Initialize CPU Backend*

Description

Creates a new CPU backend instance for graph computation.

Usage

```
ggml_backend_cpu_init()
```

Value

Backend pointer

ggml_backend_cpu_set_n_threads *Set CPU Backend Threads*

Description

Sets the number of threads for CPU backend computation.

Usage

```
ggml_backend_cpu_set_n_threads(backend, n_threads)
```

Arguments

backend	CPU backend pointer
n_threads	Number of threads

Value

NULL invisibly

`ggml_backend_dev_buffer_type` *Get the default buffer type of a device*

Description

Returns the buffer type a device allocates from by default. This is the buffer type the device itself reports as supported, so it is the natural argument to `ggml_backend_dev_supports_buft`.

Usage

```
ggml_backend_dev_buffer_type(device)
```

Arguments

`device`	External pointer to device

Details

The buffer type is owned by the backend and lives for as long as the backend does; the returned external pointer has no finalizer and must not be freed.

Value

External pointer to the buffer type, or NULL if the device has none

See Also

Other backend: `ggml_backend_buffer_clear()`, `ggml_backend_buffer_get_usage()`, `ggml_backend_buffer_is_host`
`ggml_backend_buffer_is_multi_buffer()`, `ggml_backend_buffer_reset()`, `ggml_backend_buffer_set_usage()`,
`ggml_backend_buffer_usage_any()`, `ggml_backend_buffer_usage_compute()`, `ggml_backend_buffer_usage_weigh`
`ggml_backend_buft_get_alignment()`, `ggml_backend_buft_get_max_size()`, `ggml_backend_buft_is_host()`,
`ggml_backend_buft_name()`, `ggml_backend_dev_by_name()`, `ggml_backend_dev_by_type()`,
`ggml_backend_dev_count()`, `ggml_backend_dev_description()`, `ggml_backend_dev_get()`,
`ggml_backend_dev_get_props()`, `ggml_backend_dev_host_buffer_type()`, `ggml_backend_dev_init()`,
`ggml_backend_dev_memory()`, `ggml_backend_dev_name()`, `ggml_backend_dev_offload_op()`,
`ggml_backend_dev_supports_buft()`, `ggml_backend_dev_supports_op()`, `ggml_backend_dev_type()`,
`ggml_backend_device_register()`, `ggml_backend_device_type_accel()`, `ggml_backend_device_type_cpu()`,
`ggml_backend_device_type_gpu()`, `ggml_backend_device_type_igpu()`, `ggml_backend_event_free()`,
`ggml_backend_event_new()`, `ggml_backend_event_record()`, `ggml_backend_event_synchronize()`,
`ggml_backend_event_wait()`, `ggml_backend_get_device()`, `ggml_backend_graph_compute_async()`,
`ggml_backend_graph_plan_compute()`, `ggml_backend_graph_plan_create()`, `ggml_backend_graph_plan_free()`,
`ggml_backend_init_best()`, `ggml_backend_init_by_name()`, `ggml_backend_init_by_type()`,
`ggml_backend_load()`, `ggml_backend_load_all()`, `ggml_backend_meta_device()`, `ggml_backend_multi_buffer_al`
`ggml_backend_multi_buffer_set_usage()`, `ggml_backend_reg_by_name()`, `ggml_backend_reg_count()`,
`ggml_backend_reg_dev_count()`, `ggml_backend_reg_dev_get()`, `ggml_backend_reg_get()`,
`ggml_backend_reg_name()`, `ggml_backend_register()`, `ggml_backend_synchronize()`, `ggml_backend_tensor_copy`
`ggml_backend_tensor_get_async()`, `ggml_backend_tensor_set_async()`, `ggml_backend_unload()`

Examples

```
dev <- ggml_backend_dev_get(0)
buft <- ggml_backend_dev_buffer_type(dev)
ggml_backend_buft_name(buft)
ggml_backend_dev_supports_buft(dev, buft)  # TRUE: a device supports its own
```

---

ggml_backend_dev_by_name

Get device by name

Description

Get device by name

Usage

```
ggml_backend_dev_by_name(name)
```

Arguments

name	Device name

Value

External pointer to device, or NULL if not found

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_type(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),

ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_dev_by_type *Get device by type*

Description

Get device by type

Usage

```
ggml_backend_dev_by_type(type)
```

Arguments

type	Device type (use ggml_backend_device_type_* functions)

Value

External pointer to first device of given type, or NULL if not found

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_dev_count

Get number of available devices

Description

Get number of available devices

Usage

```
ggml_backend_dev_count()
```

Value

Number of devices

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_description(), ggml_backend_dev_get(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_dev_description

Get device description

Description

Get device description

Usage

```
ggml_backend_dev_description(device)
```

Arguments

device	External pointer to device

Value

Device description

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_get(), ggml_backend_dev_get_prop
ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(),
ggml_backend_dev_name(), ggml_backend_dev_offload_op(), ggml_backend_dev_supports_buft(),
ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_dev_get *Get device by index*

Description

Get device by index

Usage

```
ggml_backend_dev_get(index)
```

Arguments

index	Device index (0-based)

Value

External pointer to device, or NULL if not found

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_dev_get_props *Get device properties*

Description

Get device properties

Usage

```
ggml_backend_dev_get_props(device)
```

Arguments

device	External pointer to device

Value

List with name, description, memory_free, memory_total, type, device_id, caps

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_host_buffer_type` *Get the host (pinned) buffer type of a device*

Description

Returns the buffer type used for pinned host memory, which allows faster host-device transfers. Many devices — including the CPU backend — have no such buffer type and return NULL.

Usage

```
ggml_backend_dev_host_buffer_type(device)
```

Arguments

`device`	External pointer to device

Details

As with ggml_backend_dev_buffer_type, the returned pointer is owned by the backend and must not be freed.

Value

External pointer to the host buffer type, or NULL if unavailable

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_init(), ggml_backend_dev_memory
ggml_backend_dev_name(), ggml_backend_dev_offload_op(), ggml_backend_dev_supports_buft(),
ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_init` *Initialize backend from device*

Description

Initialize backend from device

Usage

```
ggml_backend_dev_init(device, params = NULL)
```

Arguments

`device`	External pointer to device
`params`	Optional parameters string

Value

External pointer to backend, or NULL on failure

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(),
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_memory` *Get device memory*

Description

Get device memory

Usage

```
ggml_backend_dev_memory(device)
```

Arguments

`device`	External pointer to device

Value

Named numeric vector with 'free' and 'total' memory in bytes

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_name(), ggml_backend_dev_offload_op(), ggml_backend_dev_suppo
ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_name` *Get device name*

Description

Get device name

Usage

```
ggml_backend_dev_name(device)
```

Arguments

`device`	External pointer to device

Value

Device name

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),

ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_offload_op(), ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_offload_op` *Check if device should offload operation*

Description

Check if device should offload operation

Usage

```
ggml_backend_dev_offload_op(device, op)
```

Arguments

`device`	External pointer to device
`op`	External pointer to tensor/operation

Value

Logical indicating if operation should be offloaded

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),

ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_supports_
ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_supports_buft` *Check if device supports buffer type*

Description

Check if device supports buffer type

Usage

```
ggml_backend_dev_supports_buft(device, buft)
```

Arguments

`device`	External pointer to device
`buft`	External pointer to buffer type

Value

Logical indicating support

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),

ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_supports_op` *Check if device supports operation*

Description

Check if device supports operation

Usage

```
ggml_backend_dev_supports_op(device, op)
```

Arguments

`device`	External pointer to device
`op`	External pointer to tensor/operation

Value

Logical indicating support

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_ ggml_backend_dev_supports_buft(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),

ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_dev_type` *Get device type*

Description

Get device type

Usage

```
ggml_backend_dev_type(device)
```

Arguments

`device`	External pointer to device

Value

Device type constant

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_device_register(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_device_register *Register a device*

Description

Dynamically registers a new device in the global registry. This is an advanced function for custom backend development.

Usage

```
ggml_backend_device_register(device)
```

Arguments

device	External pointer to device

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_device_type_accel

Device type: Accelerator

Description

Device type: Accelerator

Usage

```
ggml_backend_device_type_accel()
```

Value

Integer constant for accelerator device type (e.g. BLAS, AMX)

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_device_type_cpu

Device type: CPU

Description

Device type: CPU

Usage

```
ggml_backend_device_type_cpu()
```

Value

Integer constant for CPU device type

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_gpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_device_type_gpu

Device type: GPU

Description

Device type: GPU

Usage

```
ggml_backend_device_type_gpu()
```

Value

Integer constant for GPU device type

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_device_type_igpu

Device type: Integrated GPU

Description

Device type: Integrated GPU

Usage

```
ggml_backend_device_type_igpu()
```

Value

Integer constant for integrated GPU device type

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),

ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_c
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_event_free(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_event_free` *Free event*

Description

Free event

Usage

```
ggml_backend_event_free(event)
```

Arguments

`event`	External pointer to event

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_c
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_new(),
ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(),

ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_event_new *Create new event*

Description

Create new event

Usage

```
ggml_backend_event_new(device)
```

Arguments

device	External pointer to device

Value

External pointer to event, or NULL on failure

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_c ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),

ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_event_record` *Record event*

Description

Record event

Usage

```
ggml_backend_event_record(event, backend)
```

Arguments

event	External pointer to event
backend	External pointer to backend

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host, ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh, ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_, ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u, ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_event_synchronize` *Synchronize event*

Description

Synchronize event

Usage

```
ggml_backend_event_synchronize(event)
```

Arguments

`event`	External pointer to event

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_wait(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_event_wait *Wait for event*

Description

Wait for event

Usage

```
ggml_backend_event_wait(backend, event)
```

Arguments

backend	External pointer to backend
event	External pointer to event

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_free *Free Backend*

Description

Releases resources associated with a backend.

Usage

```
ggml_backend_free(backend)
```

Arguments

backend	Backend pointer

Value

NULL invisibly

ggml_backend_get_device *Get device from backend*

Description

Get device from backend

Usage

```
ggml_backend_get_device(backend)
```

Arguments

backend	External pointer to backend

Value

External pointer to device

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_graph_compute` *Compute Graph with Backend*

Description

Executes computation graph using specified backend.

Usage

```
ggml_backend_graph_compute(backend, graph)
```

Arguments

backend	Backend pointer
graph	Graph pointer

Value

Status code (0 = success)

`ggml_backend_graph_compute_async` *Compute graph asynchronously*

Description

Starts graph computation without blocking. Use ggml_backend_synchronize() to wait for completion.

Usage

```
ggml_backend_graph_compute_async(backend, graph)
```

Arguments

backend	External pointer to backend
graph	External pointer to computation graph

Value

Integer status code (0 = success)

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_plan_compute(),
ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

Examples

```
cpu <- ggml_backend_cpu_init()
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
b <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, b)
ggml_set_f32(a, rnorm(100))
# Start async computation
status <- ggml_backend_graph_compute_async(cpu, graph)
# Do other work while computation runs...
ggml_backend_synchronize(cpu)
ggml_backend_free(cpu)
ggml_free(ctx)
```

---

ggml_backend_graph_plan_compute

Execute graph plan

Description

Execute graph plan

Usage

```
ggml_backend_graph_plan_compute(backend, plan)
```

Arguments

backend	External pointer to backend
plan	External pointer to plan

Value

Status code (0 = success)

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),

ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_graph_plan_create` *Create graph execution plan*

Description

Create graph execution plan

Usage

```
ggml_backend_graph_plan_create(backend, graph)
```

Arguments

backend	External pointer to backend
graph	External pointer to computation graph

Value

External pointer to plan, or NULL on failure

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_ ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),

ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_free(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_graph_plan_free` *Free graph execution plan*

Description

Free graph execution plan

Usage

```
ggml_backend_graph_plan_free(backend, plan)
```

Arguments

backend	External pointer to backend
plan	External pointer to plan

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_init_best(),
ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us

ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_init_best

Initialize best available backend

Description

Initialize best available backend

Usage

```
ggml_backend_init_best()
```

Value

External pointer to backend (GPU if available, otherwise CPU)

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host, ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weight, ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_, ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_us, ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_init_by_name *Initialize backend by name*

Description

Initialize backend by name

Usage

```
ggml_backend_init_by_name(name, params = NULL)
```

Arguments

name	Backend name (e.g. "CPU", "Vulkan")
params	Optional parameters string

Value

External pointer to backend, or NULL on failure

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_init_by_type *Initialize backend by type*

Description

Initialize backend by type

Usage

```
ggml_backend_init_by_type(type, params = NULL)
```

Arguments

type	Device type constant
params	Optional parameters string

Value

External pointer to backend, or NULL on failure

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_load(), ggml_backend_load_all(),
ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_u
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_load *Load backend from dynamic library*

Description

Load backend from dynamic library

Usage

```
ggml_backend_load(path)
```

Arguments

path	Path to dynamic library

Value

External pointer to registry, or NULL on failure

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(),
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_load_all *Load all available backends*

Description

Load all available backends

Usage

```
ggml_backend_load_all()
```

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_meta_device(), ggml_backend_multi_buffer_alloc_buffer(),
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_meta_device *Create a Meta Backend Device*

Description

Creates a "meta" device that wraps multiple "simple" backend devices for tensor parallelism. Each tensor is split across the wrapped devices according to the result of `split_fn`, which is called by ggml when weight buffers are allocated.

Usage

```
ggml_backend_meta_device(devs, split_fn, env = environment(split_fn))
```

Arguments

devs	A list of ggml_backend_dev_t external pointers.
split_fn	A function function(tensor_info, n_devs) returning the split state as described above.
env	An environment in which to evaluate split_fn; defaults to the function's enclosing environment.

Details

The split function is invoked with two arguments:

tensor_info a named list with fields name (character), type (integer ggml_type enum), ne (numeric vector of dimensions), op (integer op enum), flags (integer).

n_devs the number of simple devices wrapped by the meta backend.

It must return a named list with:

axis integer; one of 0..3 to split along a tensor axis, 10 for MIRRORED (full copy on each device), 11 for PARTIAL (each device has a partial sum), or 98/99 for NONE/UNKNOWN.

ne integer or numeric vector of length n_segments * n_devs giving the per-segment, per-device slice size along the split axis.

n_segments integer; usually 1, larger for fused tensors like QKV.

If split_fn errors or returns an unparseable result, the meta backend silently falls back to MIRRORED for that tensor and stops calling the callback (sticky error). This is intentional: a misbehaving callback would otherwise spray errors for every tensor in the model.

Note: with a single device this is a degenerate (no-op) configuration — useful for testing but provides no parallelism benefit. The feature is **experimental** and the API may change.

Value

External pointer to the meta ggml_backend_dev_t.

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),

ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(), ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_multi_buffer_alloc_buffer(), ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_multi_buffer_alloc_buffer` *Allocate multi-buffer*

Description

Creates a buffer that combines multiple backend buffers into one. Useful for managing memory across different backends.

Usage

```
ggml_backend_multi_buffer_alloc_buffer(buffers)
```

Arguments

`buffers`	List of backend buffer external pointers

Value

External pointer to multi-buffer

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(), ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(), ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(), ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(), ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(), ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(), ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),

ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(), ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(), ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(), ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

Examples

```
cpu <- ggml_backend_cpu_init()
ctx1 <- ggml_init(1024, no_alloc = TRUE)
ctx2 <- ggml_init(2048, no_alloc = TRUE)
a <- ggml_new_tensor_1d(ctx1, GGML_TYPE_F32, 10)
b <- ggml_new_tensor_1d(ctx2, GGML_TYPE_F32, 20)
buf1 <- ggml_backend_alloc_ctx_tensors(ctx1, cpu)
buf2 <- ggml_backend_alloc_ctx_tensors(ctx2, cpu)
multi <- ggml_backend_multi_buffer_alloc_buffer(list(buf1, buf2))
ggml_backend_buffer_free(multi)
ggml_backend_free(cpu)
ggml_free(ctx1)
ggml_free(ctx2)
```

---

ggml_backend_multi_buffer_set_usage

Set usage for all buffers in a multi-buffer

Description

Set usage for all buffers in a multi-buffer

Usage

```
ggml_backend_multi_buffer_set_usage(buffer, usage)
```

Arguments

buffer	External pointer to multi-buffer
usage	Usage constant (from ggml_backend_buffer_usage_*)

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_reg_by_name(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_name` *Get Backend Name*

Description

Returns the name of the backend (e.g., "CPU").

Usage

```
ggml_backend_name(backend)
```

Arguments

backend	Backend pointer

Value

Character string name

ggml_backend_reg_by_name *Get backend registry by name*

Description

Get backend registry by name

Usage

```
ggml_backend_reg_by_name(name)
```

Arguments

name	Registry name

Value

External pointer to registry, or NULL if not found

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_count(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_reg_count

Get number of registered backends

Description

Get number of registered backends

Usage

```
ggml_backend_reg_count()
```

Value

Number of registered backends

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_dev_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_reg_dev_count

Get number of devices in registry

Description

Get number of devices in registry

Usage

```
ggml_backend_reg_dev_count(reg)
```

Arguments

reg	External pointer to registry

Value

Number of devices

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_get(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_reg_dev_get

Get device from registry

Description

Get device from registry

Usage

```
ggml_backend_reg_dev_get(reg, index)
```

Arguments

reg	External pointer to registry
index	Device index (0-based)

Value

External pointer to device

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_get(), ggml_backend_reg_name(), ggml_backend_register(),
ggml_backend_synchronize(), ggml_backend_tensor_copy_async(), ggml_backend_tensor_get_async(),
ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_reg_get *Get backend registry by index*

Description

Get backend registry by index

Usage

```
ggml_backend_reg_get(index)
```

Arguments

index	Registry index (0-based)

Value

External pointer to registry, or NULL if not found

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_name(),
ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(),
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_reg_name *Get registry name*

Description

Get registry name

Usage

```
ggml_backend_reg_name(reg)
```

Arguments

reg	External pointer to registry

Value

Registry name

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(),
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

`ggml_backend_register` *Register a backend*

Description

Dynamically registers a new backend in the global registry. This is an advanced function for custom backend development.

Usage

```
ggml_backend_register(reg)
```

Arguments

`reg`	External pointer to backend registry

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_synchronize(), ggml_backend_tensor_copy_async(),
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_sched_alloc_graph

Allocate graph on scheduler

Description

Allocates memory for a graph across the scheduler's backends. Must be called before computing the graph.

Usage

```
ggml_backend_sched_alloc_graph(sched, graph)
```

Arguments

sched	Scheduler pointer
graph	Graph pointer

Value

Logical indicating success

ggml_backend_sched_free *Free backend scheduler*

Description

Releases resources associated with the backend scheduler.

Usage

```
ggml_backend_sched_free(sched)
```

Arguments

sched	Scheduler pointer from ggml_backend_sched_new()

Value

NULL (invisible)

Examples

```
cpu <- ggml_backend_cpu_init()
sched <- ggml_backend_sched_new(list(cpu))
ggml_backend_sched_free(sched)
ggml_backend_free(cpu)
```

ggml_backend_sched_get_backend *Get backend from scheduler*

Description

Returns a specific backend from the scheduler by index.

Usage

```
ggml_backend_sched_get_backend(sched, index = 0L)
```

Arguments

sched	Scheduler pointer
index	Backend index (0-based)

Value

Backend pointer

ggml_backend_sched_get_n_backends

Get number of backends in scheduler

Description

Returns the number of backends managed by the scheduler.

Usage

```
ggml_backend_sched_get_n_backends(sched)
```

Arguments

sched	Scheduler pointer

Value

Integer count of backends

ggml_backend_sched_get_n_copies

Get number of tensor copies

Description

Returns the number of tensor copies made in the last computed graph. Copies occur when data needs to be transferred between backends.

Usage

```
ggml_backend_sched_get_n_copies(sched)
```

Arguments

sched	Scheduler pointer

Value

Integer count of copies

ggml_backend_sched_get_n_splits

Get number of graph splits

Description

Returns the number of splits in the last computed graph. Higher numbers indicate more distribution across backends.

Usage

```
ggml_backend_sched_get_n_splits(sched)
```

Arguments

sched	Scheduler pointer

Value

Integer count of splits

ggml_backend_sched_get_tensor_backend

Get tensor backend assignment

Description

Returns which backend a tensor is assigned to.

Usage

```
ggml_backend_sched_get_tensor_backend(sched, tensor)
```

Arguments

sched	Scheduler pointer
tensor	Tensor pointer

Details

Assignments are only meaningful once the scheduler has actually placed the graph — that is, after ggml_backend_sched_graph_compute. After ggml_backend_sched_reserve alone, or for a tensor from a different graph, the assignment may not exist and this returns NULL.

Value

Backend pointer, or NULL if the tensor is not assigned to one

ggml_backend_sched_graph_compute *Compute graph using scheduler*

Description

Computes a graph by distributing work across multiple backends. This is the main function for multi-GPU computation.

Usage

```
ggml_backend_sched_graph_compute(sched, graph)
```

Arguments

sched	Scheduler pointer
graph	Graph pointer

Value

Status code (0 = success)

Examples

```
# Multi-GPU example
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 2) {
  gpu1 <- ggml_vulkan_init(0)
  gpu2 <- ggml_vulkan_init(1)
  sched <- ggml_backend_sched_new(list(gpu1, gpu2))

  ctx <- ggml_init(64 * 1024 * 1024)
  a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10000)
  b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10000)
  ggml_set_f32(a, rnorm(10000))
  ggml_set_f32(b, rnorm(10000))

  c <- ggml_add(ctx, a, b)
  graph <- ggml_build_forward_expand(ctx, c)

  # Reserve memory
  ggml_backend_sched_reserve(sched, graph)

  # Compute using both GPUs
  ggml_backend_sched_graph_compute(sched, graph)

  result <- ggml_get_f32(c)

  cat("Splits:", ggml_backend_sched_get_n_splits(sched), "\n")
  cat("Copies:", ggml_backend_sched_get_n_copies(sched), "\n")
```

```
  ggml_free(ctx)
  ggml_backend_sched_free(sched)
  ggml_vulkan_free(gpu1)
  ggml_vulkan_free(gpu2)
}
```

`ggml_backend_sched_graph_compute_async` *Compute graph asynchronously*

Description

Computes a graph asynchronously across backends. Use ggml_backend_sched_synchronize() to wait for completion.

Usage

```
ggml_backend_sched_graph_compute_async(sched, graph)
```

Arguments

`sched`	Scheduler pointer
`graph`	Graph pointer

Value

Status code (0 = success)

`ggml_backend_sched_new` *Create a new backend scheduler*

Description

Creates a scheduler that can distribute computation across multiple backends (GPUs, CPU). A CPU backend is automatically added as a fallback. Backends with lower index have higher priority.

Usage

```
ggml_backend_sched_new(backends, parallel = TRUE, graph_size = 2048)
```

Arguments

backends	List of backend pointers (from ggml_vulkan_init() or ggml_backend_cpu_init()). Note: A CPU backend is automatically added, so you only need to specify GPU backends.
parallel	Logical, whether to run backends in parallel (default: TRUE)
graph_size	Expected maximum graph size (default: 2048)

Value

Scheduler pointer

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 2) {
  # Create two GPU backends (CPU is added automatically)
  gpu1 <- ggml_vulkan_init(0)
  gpu2 <- ggml_vulkan_init(1)

  # Create scheduler with both GPUs + CPU (automatic)
  sched <- ggml_backend_sched_new(list(gpu1, gpu2), parallel = TRUE)

  # The scheduler now has 3 backends: GPU1, GPU2, CPU
  cat("Backends:", ggml_backend_sched_get_n_backends(sched), "\\n")

  # Use scheduler...

  # Cleanup
  ggml_backend_sched_free(sched)
  ggml_vulkan_free(gpu1)
  ggml_vulkan_free(gpu2)
}
```

---

ggml_backend_sched_reserve

Reserve memory for scheduler

Description

Pre-allocates memory based on a measurement graph. This should be called before using the scheduler to compute graphs.

Usage

```
ggml_backend_sched_reserve(sched, graph)
```

Arguments

sched	Scheduler pointer
graph	Graph pointer to measure memory requirements

Value

Logical indicating success

Examples

```
cpu <- ggml_backend_cpu_init()
sched <- ggml_backend_sched_new(list(cpu))
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 1000)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 1000)
c <- ggml_add(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, c)
ggml_backend_sched_reserve(sched, graph)
ggml_backend_sched_free(sched)
ggml_backend_free(cpu)
ggml_free(ctx)
```

---

ggml_backend_sched_reset

Reset scheduler

Description

Resets the scheduler, deallocating all tensors. Must be called before changing node backends or allocating a new graph.

Usage

```
ggml_backend_sched_reset(sched)
```

Arguments

sched	Scheduler pointer

Value

NULL (invisible)

ggml_backend_sched_set_tensor_backend *Set tensor backend assignment*

Description

Manually assigns a specific tensor to run on a specific backend. This overrides automatic scheduling.

Usage

```
ggml_backend_sched_set_tensor_backend(sched, tensor, backend)
```

Arguments

sched	Scheduler pointer
tensor	Tensor pointer
backend	Backend pointer to assign tensor to

Value

NULL (invisible)

ggml_backend_sched_synchronize *Synchronize scheduler*

Description

Waits for all asynchronous operations to complete.

Usage

```
ggml_backend_sched_synchronize(sched)
```

Arguments

sched	Scheduler pointer

Value

NULL (invisible)

ggml_backend_sched_trace

Trace Every Node a Scheduler Computes

Description

Turns on a per-node trace: for each node the scheduler evaluates, its name, op, shape and value summary (sum/min/max) are printed to stderr. Nodes are identified by NAME rather than index, so traces from two backends can be diffed even when their graphs enumerate nodes differently – which is what makes this useful for finding where a CPU and a GPU run diverge.

Usage

```
ggml_backend_sched_trace(sched, enable = TRUE)
```

Arguments

sched	Backend scheduler
enable	Logical; turn tracing on or off

Details

Off by default; leaving it on makes computation much slower.

Value

NULL, invisibly

ggml_backend_synchronize

Synchronize backend

Description

Synchronize backend

Usage

```
ggml_backend_synchronize(backend)
```

Arguments

backend	External pointer to backend

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_tensor_copy_async(),
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_tensor_copy_async

Copy tensor asynchronously between backends

Description

Copy tensor asynchronously between backends

Usage

```
ggml_backend_tensor_copy_async(backend_src, backend_dst, src, dst)
```

Arguments

backend_src	Source backend
backend_dst	Destination backend
src	Source tensor
dst	Destination tensor

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_get_a
ggml_backend_tensor_set_async(), ggml_backend_unload()

---

ggml_backend_tensor_get_and_sync

Backend Tensor Get and Sync

Description

Gets tensor data from a backend with synchronization.

Usage

```
ggml_backend_tensor_get_and_sync(backend, tensor, offset = 0, size)
```

Arguments

backend	Backend pointer (or NULL for CPU)
tensor	Tensor pointer
offset	Byte offset (default 0)
size	Number of bytes to read

Value

Raw vector with tensor data

ggml_backend_tensor_get_async *Get tensor data asynchronously*

Description

Get tensor data asynchronously

Usage

```
ggml_backend_tensor_get_async(backend, tensor, offset = 0, size)
```

Arguments

backend	External pointer to backend
tensor	External pointer to tensor
offset	Byte offset (default 0)
size	Number of bytes to read

Value

Numeric vector with data

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_set_async(), ggml_backend_unload()

ggml_backend_tensor_get_data

Get Tensor Data via Backend

Description

Gets tensor data using the backend API. This works with tensors allocated on any backend, not just CPU.

Usage

```
ggml_backend_tensor_get_data(tensor, offset = 0, n_elements = NULL)
```

Arguments

tensor	Tensor pointer
offset	Byte offset (default: 0)
n_elements	Number of elements to retrieve (NULL for all)

Value

R vector with tensor data

ggml_backend_tensor_get_f32_first

Get First Float from Backend Tensor

Description

Reads the first f32 element from a backend tensor.

Usage

```
ggml_backend_tensor_get_f32_first(tensor)
```

Arguments

tensor	Tensor pointer

Value

Float value

ggml_backend_tensor_set_async *Set tensor data asynchronously*

Description

Set tensor data asynchronously

Usage

```
ggml_backend_tensor_set_async(backend, tensor, data, offset = 0, size = NULL)
```

Arguments

backend	External pointer to backend
tensor	External pointer to tensor
data	Numeric or integer vector
offset	Byte offset (default 0)
size	Number of bytes to copy

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_unload()

ggml_backend_tensor_set_data *Set Tensor Data via Backend*

Description

Sets tensor data using the backend API. This works with tensors allocated on any backend, not just CPU.

Usage

```
ggml_backend_tensor_set_data(tensor, data, offset = 0)
```

Arguments

tensor	Tensor pointer
data	R vector with data to set
offset	Byte offset (default: 0)

Details

Integer and logical data are accepted for float tensors and converted element by element to float (an integer NA becomes NaN). An I32 tensor (token indices, say) takes integers or whole-valued doubles; NA, fractions and values outside the 32-bit range stop with an error rather than being truncated.

Value

No return value, called for side effects

ggml_backend_unload *Unload backend*

Description

Unload backend

Usage

```
ggml_backend_unload(reg)
```

Arguments

reg	External pointer to registry

Value

NULL invisibly

See Also

Other backend: ggml_backend_buffer_clear(), ggml_backend_buffer_get_usage(), ggml_backend_buffer_is_host
ggml_backend_buffer_is_multi_buffer(), ggml_backend_buffer_reset(), ggml_backend_buffer_set_usage(),
ggml_backend_buffer_usage_any(), ggml_backend_buffer_usage_compute(), ggml_backend_buffer_usage_weigh
ggml_backend_buft_get_alignment(), ggml_backend_buft_get_max_size(), ggml_backend_buft_is_host(),
ggml_backend_buft_name(), ggml_backend_dev_buffer_type(), ggml_backend_dev_by_name(),
ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(),
ggml_backend_dev_get(), ggml_backend_dev_get_props(), ggml_backend_dev_host_buffer_type(),
ggml_backend_dev_init(), ggml_backend_dev_memory(), ggml_backend_dev_name(), ggml_backend_dev_offload_
ggml_backend_dev_supports_buft(), ggml_backend_dev_supports_op(), ggml_backend_dev_type(),
ggml_backend_device_register(), ggml_backend_device_type_accel(), ggml_backend_device_type_cpu(),
ggml_backend_device_type_gpu(), ggml_backend_device_type_igpu(), ggml_backend_event_free(),
ggml_backend_event_new(), ggml_backend_event_record(), ggml_backend_event_synchronize(),
ggml_backend_event_wait(), ggml_backend_get_device(), ggml_backend_graph_compute_async(),
ggml_backend_graph_plan_compute(), ggml_backend_graph_plan_create(), ggml_backend_graph_plan_free(),
ggml_backend_init_best(), ggml_backend_init_by_name(), ggml_backend_init_by_type(),
ggml_backend_load(), ggml_backend_load_all(), ggml_backend_meta_device(), ggml_backend_multi_buffer_al
ggml_backend_multi_buffer_set_usage(), ggml_backend_reg_by_name(), ggml_backend_reg_count(),
ggml_backend_reg_dev_count(), ggml_backend_reg_dev_get(), ggml_backend_reg_get(),
ggml_backend_reg_name(), ggml_backend_register(), ggml_backend_synchronize(), ggml_backend_tensor_copy_
ggml_backend_tensor_get_async(), ggml_backend_tensor_set_async()

---

ggml_batch_norm *Create a Batch Normalization Layer Object*

Description

Create a Batch Normalization Layer Object

Usage

```
ggml_batch_norm(eps = 1e-05, name = NULL, trainable = TRUE)
```

Arguments

eps	Small constant for numerical stability (default 1e-5).
name	Optional character name.
trainable	Logical.

Value

A ggml_layer object.

ggml_blck_size *Get Block Size*

Description

Returns the block size for a GGML type. Quantized types process data in blocks (e.g., 32 elements for Q4_0).

Usage

```
ggml_blck_size(type)
```

Arguments

type	GGML type constant

Value

Integer block size

See Also

Other type_system: ggml_ftype_to_ggml_type(), ggml_is_quantized(), ggml_type_name(), ggml_type_sizef()

Examples

```
ggml_blck_size(GGML_TYPE_F32)  # 1
ggml_blck_size(GGML_TYPE_Q4_0) # 32
```

ggml_build_backward_expand *Add the Backward Pass to a Computation Graph*

Description

Extends graph in place with the nodes that compute gradients of its output with respect to every tensor marked by ggml_set_param. After computing the graph, read a gradient with ggml_graph_get_grad.

Usage

```
ggml_build_backward_expand(ctx, graph)
```

Arguments

ctx	GGML context
graph	Graph from ggml_build_forward_expand_grads

Details

The graph must come from ggml_build_forward_expand_grads: ggml_build_forward_expand produces one without gradient storage, and that is rejected rather than allowed to abort inside ggml.

Value

NULL, invisibly; graph is modified in place

The full sequence

Every step is required; skipping one gives either an error or – for the last – gradients that are silently all zero.

1. ggml_set_param on each tensor to differentiate
2. build the forward ops
3. ggml_set_loss on the output
4. ggml_build_forward_expand_grads
5. ggml_build_backward_expand()
6. ggml_graph_reset – seeds d(loss)/d(loss) = 1
7. ggml_graph_compute
8. ggml_graph_get_grad to read a gradient

Which ops are differentiable

Gradients flow only through operations that implement a backward pass. Most do, but not all – ggml_clamp is a notable exception, and among the state-space ops only ggml_ssm_conv is differentiable.

See Also

ggml_build_forward_expand_grads, ggml_graph_get_grad, ggml_set_param

---

ggml_build_forward_expand

Build forward expand

Description

Builds a computation graph from the output tensor, expanding backwards to include all dependencies.

Creates a computation graph by expanding backwards from the output tensor

Usage

```
ggml_build_forward_expand(ctx, tensor)

ggml_build_forward_expand(ctx, tensor)
```

Arguments

ctx	GGML context
tensor	Output tensor of the computation

Value

Graph pointer

Graph object (external pointer)

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(5, 4, 3, 2, 1))
result <- ggml_add(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)


ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_set_f32(a, 1:10)
ggml_set_f32(b, 11:20)
c <- ggml_add(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, c)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(c)
ggml_free(ctx)
```

---

ggml_build_forward_expand_grads

Build a Forward Graph That Can Be Differentiated

Description

Like ggml_build_forward_expand, but the graph also reserves storage for gradients, which is what ggml_build_backward_expand requires. A graph from the plain forward builder carries none, so passing it to the backward builder is an error.

Usage

```
ggml_build_forward_expand_grads(ctx, tensor, graph_size = 2048L)
```

Arguments

ctx	GGML context
tensor	Output tensor of the computation (usually a scalar loss)
graph_size	Node capacity of the graph (default 2048)

Details

Mark the tensors to differentiate with respect to using ggml_set_param *before* building the graph.

Value

Graph object (external pointer)

Graph size

graph_size is the node capacity. The backward pass appends nodes to the forward ones, so it needs more room than a forward-only graph: roughly three times the forward node count is a safe starting point. Too small a value fails while the graph is being built, not silently.

See Also

ggml_build_backward_expand, ggml_graph_get_grad

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
ggml_set_f32(b, c(5, 6, 7, 8))
ggml_set_param(a)
loss  <- ggml_sum(ctx, ggml_mul(ctx, a, b))
ggml_set_loss(loss)
graph <- ggml_build_forward_expand_grads(ctx, loss)
ggml_build_backward_expand(ctx, graph)
ggml_graph_reset(graph)     # seeds d(loss)/d(loss) = 1
ggml_graph_compute(ctx, graph)
# d(sum(a*b))/da == b
ggml_get_f32(ggml_graph_get_grad(graph, a))
ggml_free(ctx)
```

ggml_callback_early_stopping *Early stopping callback*

Description

Stops training when the monitored metric does not improve.

Usage

```
ggml_callback_early_stopping(
  monitor = "val_loss",
  patience = 5,
  min_delta = 0,
  mode = "auto"
)
```

Arguments

monitor	Metric to monitor: "val_loss", "val_accuracy", "train_loss", "train_accuracy"
patience	Number of epochs with no improvement before stopping
min_delta	Minimum change to qualify as improvement
mode	"min" (lower is better) or "max" (higher is better). "auto" infers from monitor name.

Value

List with on_epoch_end function

See Also

Other callbacks: ggml_schedule_cosine_decay(), ggml_schedule_reduce_on_plateau(), ggml_schedule_step_deca

ggml_can_repeat *Check If Tensor Can Be Repeated*

Description

Check if tensor a can be repeated (broadcast) to match tensor b. Used for broadcasting operations.

Usage

```
ggml_can_repeat(a, b)
```

Arguments

a	Source tensor (smaller)
b	Target tensor (larger or same size)

Value

Logical indicating if a can be repeated to match b

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_2(), ggml_is_contiguous_channels(), ggml_is_contiguous_rows(), ggml_is_contiguously_allocated()

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
b <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 8)
ggml_can_repeat(a, b)  # TRUE - a can broadcast along dim 1
ggml_free(ctx)
```

---

ggml_cast *Cast a Tensor to Another Type*

Description

Returns a tensor with the same shape as a but of the given type. Unlike ggml_cpy there is no need to allocate the destination tensor first. Some ggml operations require a specific type – for example ggml_conv_1d builds its im2col in F16 and so needs an F16 kernel – and this is the direct way to supply one.

Usage

```
ggml_cast(ctx, a, type)
```

Arguments

ctx	GGML context
a	Source tensor
type	Target type constant (e.g. GGML_TYPE_F16)

Value

Tensor of the requested type with a's shape

See Also

ggml_cast_numeric, ggml_cpy

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
a16 <- ggml_cast(ctx, a, GGML_TYPE_F16)
ggml_free(ctx)
```

---

ggml_cast_numeric *Numeric Type Conversion*

Description

Converts values numerically (truncating or rounding) rather than reinterpreting the bits as ggml_cast does. Supports F32<->I32, F32<->I16, F32<->I8, F16<->F32 and BF16<->F32.

Usage

```
ggml_cast_numeric(ctx, a, type)
```

Arguments

ctx	GGML context
a	Source tensor
type	Target type constant (e.g. GGML_TYPE_I32)

Value

Tensor of the requested type with a's shape

See Also

ggml_cast

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
i <- ggml_cast_numeric(ctx, a, GGML_TYPE_I32)
ggml_free(ctx)
```

ggml_ceil *Ceiling (Graph)*

Description

Creates a graph node for element-wise ceiling: ceil(x)

Usage

```
ggml_ceil(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the ceil operation

ggml_ceil_inplace *Ceiling In-place (Graph)*

Description

Creates a graph node for in-place element-wise ceiling.

Usage

```
ggml_ceil_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with ceiling values

ggml_clamp *Clamp (Graph)*

Description

Creates a graph node for clamping values to a range: clamp(x, min, max)

Usage

```
ggml_clamp(ctx, a, min_val, max_val)
```

Arguments

ctx	GGML context
a	Input tensor
min_val	Minimum value
max_val	Maximum value

Value

Tensor with values clamped to [min_val, max_val]

Inference only

GGML_OP_CLAMP has no backward pass, and the node it builds is a view of its input. A graph containing it therefore cannot be differentiated: the backward pass aborts with *"inplace operations are currently not supported"* rather than returning an error. This matters inside ggml_layer_custom, whose gradients flow only through operations that implement a backward pass.

For a trainable equivalent, express the clamp with operations that do: `min(x, hi)` is `x - ggml_relu(ctx, x - hi)` and `max(x, lo)` is `lo + ggml_relu(ctx, x - lo)`.

See Also

ggml_relu for the differentiable formulation above.

ggml_compile.ggml_functional_model
Compile a Sequential Model

Description

Configures the model for training: infers shapes, creates backend. Weight tensors are created at training time when batch_size is known.

Usage

```
## S3 method for class 'ggml_functional_model'
ggml_compile(
  model,
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"),
  backend = "auto",
  loss_weights = NULL,
  weight_decay = 0,
  predict_precision = "default"
)

ggml_compile(
  model,
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"),
  backend = "auto",
  loss_weights = NULL,
  weight_decay = 0,
  predict_precision = "default"
)

## S3 method for class 'ggml_sequential_model'
ggml_compile(
  model,
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"),
  backend = "auto",
  loss_weights = NULL,
  weight_decay = 0,
  predict_precision = "default"
)
```

Arguments

model	A ggml_sequential_model object
optimizer	Optimizer name: "adam" or "sgd"
loss	Loss function name: "categorical_crossentropy" (expects a softmax output), "mse", "mae", "huber" (smooth L1, delta 1) or "binary_crossentropy" (expects a sigmoid output, one independent label per unit). "mean_squared_error", "mean_absolute_error", "huber_loss" and "binary_cross_entropy" are accepted as aliases. For a multi-output functional model, a list or character vector with one loss per output – named after the output layers, or positional: loss = list(policy = "categorical_crossentropy", value = "mse").
metrics	Character vector of metrics to report from ggml_evaluate. "accuracy" is reported for multi-class outputs whether or not it is requested; "mae", "mse" and "rmse" (and the long forms "mean_absolute_error", "mean_squared_error") are computed only when asked for. Any other name is dropped with a warning.
backend	Backend to use: "auto" (GPU if available, else CPU), "cpu", or "vulkan"
loss_weights	Optional per-output weights for a multi-output model; the optimized total is sum(loss_weights[i] * loss_i). Named after the output layers or positional. Default: all outputs weighted 1.
weight_decay	Decoupled weight decay for the optimizer, in [0, 1]; 0 (the default) turns it off. It is applied to trainable weights only: layers frozen with ggml_freeze_weights are not decayed.
predict_precision	Matrix-multiply precision for prediction: "default" (as ggml builds it; f16 tiles on an fp16 Vulkan device) or "f32". Used by ggml_predict, a trainer's $predict() and its snapshots unless a call gives its own matmul_precision; training is unaffected. See the precision section of ggml_predict.

Value

The compiled model (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_max_pooling_2d(c(2, 2)) |>
  ggml_layer_flatten() |>
  ggml_layer_dense(10, activation = "softmax")
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")
```

ggml_concat *Concatenate Tensors (Graph)*

Description

Concatenates two tensors along a specified dimension. CRITICAL for KV-cache operations in transformers.

Usage

```
ggml_concat(ctx, a, b, dim = 0)
```

Arguments

ctx	GGML context
a	First tensor
b	Second tensor (must match a in all dimensions except the concat dim)
dim	Dimension along which to concatenate (0-3)

Value

Concatenated tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 3)
b <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 2)
ggml_set_f32(a, rnorm(12))
ggml_set_f32(b, rnorm(8))
# Concatenate along dimension 1: result is 4x5
c <- ggml_concat(ctx, a, b, 1)
graph <- ggml_build_forward_expand(ctx, c)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_cont *Make Contiguous (Graph)*

Description

Makes a tensor contiguous in memory. Required after permute/transpose before some operations.

Usage

```
ggml_cont(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Contiguous tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 4)
ggml_set_f32(a, 1:12)
transposed <- ggml_transpose(ctx, a)
contiguous <- ggml_cont(ctx, transposed)
ggml_free(ctx)
```

---

ggml_conv_1d *1D Convolution (Graph)*

Description

Applies 1D convolution to input data.

Usage

```
ggml_conv_1d(ctx, a, b, s0 = 1L, p0 = 0L, d0 = 1L)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0	Stride (default 1)
p0	Padding (default 0)
d0	Dilation (default 1)

Value

Convolved tensor

ggml_conv_1d_dw *Depthwise 1D Convolution (Graph)*

Description

Applies depthwise 1D convolution: each input channel is convolved with its own kernel.

Usage

```
ggml_conv_1d_dw(ctx, a, b, s0 = 1L, p0 = 0L, d0 = 1L)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0	Stride (default 1)
p0	Padding (default 0)
d0	Dilation (default 1)

Value

Convolved tensor

ggml_conv_1d_layer *Create a Reusable Conv1D Layer Object*

Description

Builds a *layer object* to be applied with ggml_apply, which is how a single set of convolution weights is shared between several inputs. This is the convolution counterpart of ggml_dense.

Usage

```
ggml_conv_1d_layer(
  filters,
  kernel_size,
  activation = NULL,
  strides = 1L,
  padding = "valid",
  name = NULL,
  trainable = TRUE
)
```

Arguments

filters	Number of output filters.
kernel_size	Integer kernel size.
activation	Activation function name or NULL.
strides	Integer stride (default 1).
padding	"valid" or "same".
name	Optional character name.
trainable	Logical.

Details

For the ordinary pipe-style form that appends a conv layer to a graph (`x |> ggml_layer_conv_1d(...)`), see ggml_layer_conv_1d.

Value

A ggml_layer object.

See Also

ggml_apply, ggml_dense

Examples

```
# One convolution shared between two sequence inputs.
shared <- ggml_conv_1d_layer(filters = 4L, kernel_size = 3L)
x1 <- ggml_input(shape = c(16L, 2L))
x2 <- ggml_input(shape = c(16L, 2L))
o1 <- ggml_apply(x1, shared)
o2 <- ggml_apply(x2, shared)
stopifnot(identical(o1$layer_id, o2$layer_id))   # same weights
```

---

ggml_conv_2d *2D Convolution (Graph)*

Description

Applies 2D convolution to input data.

Usage

```
ggml_conv_2d(ctx, a, b, s0 = 1L, s1 = 1L, p0 = 0L, p1 = 0L, d0 = 1L, d1 = 1L)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor [KW, KH, IC, OC]
b	Input data tensor [W, H, C, N]
s0	Stride dimension 0 (default 1)
s1	Stride dimension 1 (default 1)
p0	Padding dimension 0 (default 0)
p1	Padding dimension 1 (default 0)
d0	Dilation dimension 0 (default 1)
d1	Dilation dimension 1 (default 1)

Value

Convolved tensor

---

ggml_conv_2d_direct *Direct 2D Convolution (Graph)*

Description

Applies 2D convolution using the direct algorithm (no im2col).

Usage

```
ggml_conv_2d_direct(
  ctx,
  a,
  b,
  s0 = 1L,
  s1 = 1L,
  p0 = 0L,
  p1 = 0L,
  d0 = 1L,
  d1 = 1L
)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0	Stride dimension 0 (default 1)
s1	Stride dimension 1 (default 1)

p0	Padding dimension 0 (default 0)
p1	Padding dimension 1 (default 0)
d0	Dilation dimension 0 (default 1)
d1	Dilation dimension 1 (default 1)

Value

Convolved tensor

`ggml_conv_2d_dw` *Depthwise 2D Convolution (Graph)*

Description

Applies depthwise 2D convolution: each input channel is convolved with its own kernel. Uses the im2col-based path.

Usage

```
ggml_conv_2d_dw(
  ctx,
  a,
  b,
  s0 = 1L,
  s1 = 1L,
  p0 = 0L,
  p1 = 0L,
  d0 = 1L,
  d1 = 1L
)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0, s1	Strides along dim 0 and 1 (default 1)
p0, p1	Padding along dim 0 and 1 (default 0)
d0, d1	Dilation along dim 0 and 1 (default 1)

Value

Convolved tensor

ggml_conv_2d_dw_direct *Depthwise 2D Convolution, direct (Graph)*

Description

Direct depthwise 2D convolution without an explicit im2col intermediate.

Usage

```
ggml_conv_2d_dw_direct(
  ctx,
  a,
  b,
  s0 = 1L,
  s1 = 1L,
  p0 = 0L,
  p1 = 0L,
  d0 = 1L,
  d1 = 1L
)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0, s1	Strides along dim 0 and 1 (default 1)
p0, p1	Padding along dim 0 and 1 (default 0)
d0, d1	Dilation along dim 0 and 1 (default 1)

Value

Convolved tensor

ggml_conv_2d_layer *Create a Reusable Conv2D Layer Object*

Description

Builds a *layer object* to be applied with ggml_apply, which is how a single set of convolution weights is shared between several inputs. This is the convolution counterpart of ggml_dense.

Usage

```
ggml_conv_2d_layer(
  filters,
  kernel_size,
  activation = NULL,
  strides = c(1L, 1L),
  padding = "valid",
  name = NULL,
  trainable = TRUE
)
```

Arguments

filters	Number of output filters.
kernel_size	Integer or length-2 integer vector.
activation	Activation function name or NULL.
strides	Integer or length-2 integer vector (default 1).
padding	"valid" or "same".
name	Optional character name.
trainable	Logical.

Details

For the ordinary pipe-style form that appends a conv layer to a graph (`x |> ggml_layer_conv_2d(...)`), see ggml_layer_conv_2d.

Value

A ggml_layer object.

See Also

ggml_apply, ggml_dense

Examples

```
# One convolution shared between two inputs.
shared <- ggml_conv_2d_layer(filters = 4L, kernel_size = 3L, activation = "relu")
x1 <- ggml_input(shape = c(8L, 8L, 1L))
x2 <- ggml_input(shape = c(8L, 8L, 1L))
o1 <- ggml_apply(x1, shared)
o2 <- ggml_apply(x2, shared)
stopifnot(identical(o1$layer_id, o2$layer_id))   # same weights
```

ggml_conv_transpose_1d

Transposed 1D Convolution (Graph)

Description

Applies transposed 1D convolution (deconvolution) to input data.

Usage

```
ggml_conv_transpose_1d(ctx, a, b, s0 = 1L, p0 = 0L, d0 = 1L)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0	Stride (default 1)
p0	Padding (default 0)
d0	Dilation (default 1)

Value

Transposed convolved tensor

ggml_conv_transpose_2d_p0

Transposed 2D Convolution, zero padding (Graph)

Description

Applies transposed 2D convolution (deconvolution) with zero padding.

Usage

```
ggml_conv_transpose_2d_p0(ctx, a, b, stride = 1L)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
stride	Stride (default 1)

Value

Transposed convolved tensor

---

ggml_cos *Cosine (Graph)*

Description

Creates a graph node for element-wise cosine: cos(x)

Usage

```
ggml_cos(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the cos operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(0, pi/3, pi/2, pi))
result <- ggml_cos(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [1, 0.5, 0, -1]
ggml_free(ctx)
```

---

ggml_count_equal *Count Equal Elements (Graph)*

Description

Creates a graph node that counts equal elements between two tensors. Useful for accuracy computation.

Usage

```
ggml_count_equal(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor
b	Second tensor (same shape as a)

Value

Tensor containing the count of equal elements

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_can_repeat(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_2(), ggml_is_contiguous_channels(), ggml_is_contiguous_rows(ggml_is_contiguously_allocated()

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
pred <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 100)
labels <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 100)
# ... set values ...
correct <- ggml_count_equal(ctx, pred, labels)
graph <- ggml_build_forward_expand(ctx, correct)
ggml_graph_compute(ctx, graph)
# correct now contains count of matching elements
ggml_free(ctx)
```

---

ggml_cpu_add *Element-wise Addition (CPU Direct)*

Description

Performs element-wise addition of two tensors using direct CPU computation. Returns the result as an R numeric vector. Does NOT use computation graphs.

Usage

```
ggml_cpu_add(a, b)
```

Arguments

a	First tensor (must be F32 type)
b	Second tensor (must be F32 type, same size as a)

Value

Numeric vector containing the element-wise sum

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(5, 4, 3, 2, 1))
ggml_cpu_add(a, b)
ggml_free(ctx)
```

---

ggml_cpu_features *Get All CPU Features*

Description

Returns a named list of all CPU feature detection results. Useful for diagnostics and optimizing computation.

Usage

```
ggml_cpu_features()
```

Value

Named list with feature names and logical values

See Also

Other cpu_features: ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

Examples

```
features <- ggml_cpu_features()
print(features)
# On typical x86-64: sse3=TRUE, avx=TRUE, avx2=TRUE, ...
```

ggml_cpu_get_rvv_vlen *Get RISC-V Vector Length*

Description

Returns the RISC-V RVV vector length in bytes (0 if not supported).

Usage

```
ggml_cpu_get_rvv_vlen()
```

Value

Integer vector length in bytes

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

ggml_cpu_get_sve_cnt *Get SVE Vector Length (ARM)*

Description

Returns the SVE vector length in bytes (0 if not supported).

Usage

```
ggml_cpu_get_sve_cnt()
```

Value

Integer vector length in bytes

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

---

ggml_cpu_has_amx_int8 *CPU Feature Detection - AMX INT8*

Description

Check if the CPU supports AMX INT8 (Advanced Matrix Extensions). AMX provides hardware acceleration for matrix operations on Intel CPUs.

Usage

```
ggml_cpu_has_amx_int8()
```

Value

Logical indicating AMX INT8 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

---

ggml_cpu_has_arm_fma *CPU Feature Detection - ARM FMA*

Description

Check if the CPU supports ARM FMA (Fused Multiply-Add).

Usage

```
ggml_cpu_has_arm_fma()
```

Value

Logical indicating ARM FMA support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

---

ggml_cpu_has_avx *CPU Feature Detection - AVX*

Description

Check if the CPU supports AVX instructions.

Usage

```
ggml_cpu_has_avx()
```

Value

Logical indicating AVX support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

ggml_cpu_has_avx_vnni *CPU Feature Detection - AVX-VNNI*

Description

Check if the CPU supports AVX-VNNI instructions.

Usage

```
ggml_cpu_has_avx_vnni()
```

Value

Logical indicating AVX-VNNI support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_avx2 *CPU Feature Detection - AVX2*

Description

Check if the CPU supports AVX2 instructions. AVX2 provides 256-bit SIMD operations for faster matrix math.

Usage

```
ggml_cpu_has_avx2()
```

Value

Logical indicating AVX2 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

---

ggml_cpu_has_avx512 *CPU Feature Detection - AVX-512*

Description

Check if the CPU supports AVX-512 instructions. AVX-512 provides 512-bit SIMD for maximum throughput.

Usage

```
ggml_cpu_has_avx512()
```

Value

Logical indicating AVX-512 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd

---

ggml_cpu_has_avx512_bf16 *CPU Feature Detection - AVX-512 BF16*

Description

Check if the CPU supports AVX-512 BF16 (bfloat16) instructions.

Usage

```
ggml_cpu_has_avx512_bf16()
```

Value

Logical indicating AVX-512 BF16 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_avx512_vbmi

CPU Feature Detection - AVX-512 VBMI

Description

Check if the CPU supports AVX-512 VBMI instructions.

Usage

```
ggml_cpu_has_avx512_vbmi()
```

Value

Logical indicating AVX-512 VBMI support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_avx512_vnni *CPU Feature Detection - AVX-512 VNNI*

Description

Check if the CPU supports AVX-512 VNNI instructions. VNNI accelerates neural network inference with int8/int16 dot products.

Usage

```
ggml_cpu_has_avx512_vnni()
```

Value

Logical indicating AVX-512 VNNI support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_bmi2 *CPU Feature Detection - BMI2*

Description

Check if the CPU supports BMI2 (Bit Manipulation Instructions 2).

Usage

```
ggml_cpu_has_bmi2()
```

Value

Logical indicating BMI2 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_dotprod *CPU Feature Detection - Dot Product (ARM)*

Description

Check if the CPU supports ARM dot product instructions. Accelerates int8 matrix multiplication common in quantized models.

Usage

```
ggml_cpu_has_dotprod()
```

Value

Logical indicating dot product support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_f16c *CPU Feature Detection - F16C*

Description

Check if the CPU supports F16C instructions for float16 conversion.

Usage

```
ggml_cpu_has_f16c()
```

Value

Logical indicating F16C support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_fma *CPU Feature Detection - FMA*

Description

Check if the CPU supports FMA (Fused Multiply-Add) instructions. FMA allows matrix operations to run faster by combining operations.

Usage

```
ggml_cpu_has_fma()
```

Value

Logical indicating FMA support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_fp16_va *CPU Feature Detection - FP16 Vector Arithmetic (ARM)*

Description

Check if the CPU supports ARM half-precision FP16 vector arithmetic.

Usage

```
ggml_cpu_has_fp16_va()
```

Value

Logical indicating FP16 VA support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_llamafile *CPU Feature Detection - Llamafile*

Description

Check if llamafile optimizations are available.

Usage

```
ggml_cpu_has_llamafile()
```

Value

Logical indicating llamafile support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_matmul_int8 *CPU Feature Detection - INT8 Matrix Multiply (ARM)*

Description

Check if the CPU supports ARM INT8 matrix multiplication.

Usage

```
ggml_cpu_has_matmul_int8()
```

Value

Logical indicating INT8 MATMUL support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_neon *CPU Feature Detection - NEON (ARM)*

Description

Check if the CPU supports ARM NEON instructions. NEON is ARM's SIMD extension for vectorized operations.

Usage

```
ggml_cpu_has_neon()
```

Value

Logical indicating NEON support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_riscv_v *CPU Feature Detection - RISC-V Vector*

Description

Check if the CPU supports RISC-V Vector extension.

Usage

```
ggml_cpu_has_riscv_v()
```

Value

Logical indicating RISC-V V support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_sme *CPU Feature Detection - SME (ARM)*

Description

Check if the CPU supports ARM SME (Scalable Matrix Extension).

Usage

```
ggml_cpu_has_sme()
```

Value

Logical indicating SME support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_sse3 *CPU Feature Detection - SSE3*

Description

Check if the CPU supports SSE3 instructions.

Usage

```
ggml_cpu_has_sse3()
```

Value

Logical indicating SSE3 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

Examples

```
ggml_cpu_has_sse3()
```

---

ggml_cpu_has_ssse3 *CPU Feature Detection - SSSE3*

Description

Check if the CPU supports SSSE3 instructions.

Usage

```
ggml_cpu_has_ssse3()
```

Value

Logical indicating SSSE3 support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_sve *CPU Feature Detection - SVE (ARM)*

Description

Check if the CPU supports ARM SVE (Scalable Vector Extension).

Usage

```
ggml_cpu_has_sve()
```

Value

Logical indicating SVE support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

ggml_cpu_has_vsx *CPU Feature Detection - VSX (PowerPC)*

Description

Check if the CPU supports PowerPC VSX instructions.

Usage

```
ggml_cpu_has_vsx()
```

Value

Logical indicating VSX support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vxe(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_vxe *CPU Feature Detection - VXE (IBM z/Architecture)*

Description

Check if the CPU supports IBM z/Architecture VXE instructions.

Usage

```
ggml_cpu_has_vxe()
```

Value

Logical indicating VXE support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_wasm_simd()

---

ggml_cpu_has_wasm_simd *CPU Feature Detection - WebAssembly SIMD*

Description

Check if the CPU/environment supports WebAssembly SIMD.

Usage

```
ggml_cpu_has_wasm_simd()
```

Value

Logical indicating WASM SIMD support

See Also

Other cpu_features: ggml_cpu_features(), ggml_cpu_get_rvv_vlen(), ggml_cpu_get_sve_cnt(), ggml_cpu_has_amx_int8(), ggml_cpu_has_arm_fma(), ggml_cpu_has_avx(), ggml_cpu_has_avx2(), ggml_cpu_has_avx512(), ggml_cpu_has_avx512_bf16(), ggml_cpu_has_avx512_vbmi(), ggml_cpu_has_avx512_vnni(), ggml_cpu_has_avx_vnni(), ggml_cpu_has_bmi2(), ggml_cpu_has_dotprod(), ggml_cpu_has_f16c(), ggml_cpu_has_fma(), ggml_cpu_has_fp16_va(), ggml_cpu_has_llamafile(), ggml_cpu_has_matmul_int8(), ggml_cpu_has_neon(), ggml_cpu_has_riscv_v(), ggml_cpu_has_sme(), ggml_cpu_has_sse3(), ggml_cpu_has_ssse3(), ggml_cpu_has_sve(), ggml_cpu_has_vsx(), ggml_cpu_has_vxe()

---

ggml_cpu_mul *Element-wise Multiplication (CPU Direct)*

Description

Performs element-wise multiplication of two tensors using direct CPU computation. Returns the result as an R numeric vector. Does NOT use computation graphs.

Usage

```
ggml_cpu_mul(a, b)
```

Arguments

a	First tensor (must be F32 type)
b	Second tensor (must be F32 type, same size as a)

Value

Numeric vector containing the element-wise product

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(2, 2, 2, 2, 2))
ggml_cpu_mul(a, b)
ggml_free(ctx)
```

ggml_cpy *Copy Tensor with Type Conversion (Graph)*

Description

Copies tensor a into tensor b, performing type conversion if needed. The tensors must have the same number of elements. CRITICAL for type casting operations (e.g., F32 to F16).

Usage

```
ggml_cpy(ctx, a, b)
```

Arguments

ctx	GGML context
a	Source tensor
b	Destination tensor (defines output type and shape)

Value

Tensor representing the copy operation (returns b with a's data)

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Create F32 tensor
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
ggml_set_f32(a, rnorm(100))
# Create F16 tensor for output
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F16, 100)
# Copy with F32 -> F16 conversion
result <- ggml_cpy(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_cross_entropy_loss *Cross-Entropy Loss (Graph)*

Description

Creates a graph node computing the cross-entropy loss between logits a and labels b. The op applies its own softmax, so a must be logits, not probabilities.

Usage

```
ggml_cross_entropy_loss(ctx, a, b)
```

Arguments

ctx	GGML context
a	Logits
b	Labels, same shape as a

Details

Differentiable: the backward pass is wired into the graph, so this can be used as a training loss in a hand-built graph.

Value

A scalar tensor holding the loss

See Also

ggml_cross_entropy_loss_back

---

ggml_cross_entropy_loss_back

Cross-Entropy Loss Backward (Graph)

Description

Gradient node for ggml_cross_entropy_loss.

Usage

```
ggml_cross_entropy_loss_back(ctx, a, b, c)
```

Arguments

ctx	GGML context
a	Upstream gradient
b	Logits from the forward pass
c	Labels from the forward pass

Details

ggml_build_backward_expand() inserts this automatically, so it is rarely needed directly; it is exposed for hand-built backward graphs.

Value

Gradient w.r.t. the logits

See Also

ggml_cross_entropy_loss

---

ggml_crossprod *GPU cross product (drop-in for* crossprod*)*

Description

Computes t(A) %*% B (or t(A) %*% A when B is NULL) on the Vulkan GPU, returning an R matrix, with a CPU fallback. See ggml_matmul for the precision and dispatch notes.

Usage

```
ggml_crossprod(A, B = NULL, device = "auto", prec = "f32")
```

Arguments

A	A numeric matrix.
B	A numeric matrix with nrow(B) == nrow(A), or NULL (default) for t(A) %*% A.
device, prec	See ggml_matmul.

Value

t(A) %*% B as a numeric matrix.

---

ggml_cumsum *Cumulative Sum (Graph)*

Description

Creates a graph node computing the cumulative sum along the first dimension.

Usage

```
ggml_cumsum(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor of the same shape holding the running sum

ggml_custom *Custom Operation Node*

Description

Creates a graph node whose value is computed by a C kernel registered under `name`. The kernel is looked up in ggmlR's custom-op registry; supplying an unregistered name is an error, listing the names that are available.

Usage

```
ggml_custom(
  ctx,
  name,
  args = NULL,
  ne,
  type = GGML_TYPE_F32,
  n_tasks = GGML_N_TASKS_MAX
)
```

Arguments

`ctx`	GGML context
`name`	Name of a registered custom op (character scalar)
`args`	List of input tensors, or NULL for none. At most 9 tensors.
`ne`	Output shape as a numeric vector of up to 4 dimensions, innermost first (ggml column-major order). Missing dimensions default to 1.
`type`	Output data type (default `GGML_TYPE_F32`)
`n_tasks`	Number of threads the kernel may be split across, or `GGML_N_TASKS_MAX` (-1) to use as many as the backend allows. Use 1 for a kernel that is not thread-safe.

Details

Kernels are registered from C, not from R. A package that links against ggmlR registers its kernels at load time via the `ggmlR_register_custom_op` callable; see `system.file("include", "ggmlR.h", package = "ggmlR")` for the C-side contract and a worked example. R callbacks are not supported: the kernel may run off the main thread, where calling into R is unsafe.

The output tensor is newly allocated with the given type and shape. Input tensors passed in `args` become the node's sources and are visible to the kernel as `dst->src[0]`, `dst->src[1]`, and so on.

Value

Tensor pointer for the custom node

Backend

Custom ops run on the CPU backend only, because the kernel is a host function pointer that a GPU backend cannot execute. Computing a graph that contains a custom node raises an error when the target backend is not the CPU, or when a scheduler has no CPU backend to fall back to. Keep custom nodes out of graphs you intend to run entirely on Vulkan.

See Also

ggml_custom_inplace, ggml_custom_ops

Examples

```
# "row_median" is one of ggmlR's built-in kernels; see ggml_custom_ops().
ggml_set_n_threads(1L)
ctx <- ggml_init(16 * 1024 * 1024)
x <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 5, 2)
ggml_set_f32(x, c(5, 1, 3, 2, 4,
                  9, 7, 8, 6, 10))

# One median per row: the output collapses ne[1] to 1.
med <- ggml_custom(ctx, "row_median", args = list(x), ne = c(1, 2))

graph <- ggml_build_forward_expand(ctx, med)
ggml_graph_compute(ctx, graph)
ggml_get_f32(med)   # 3, 8
ggml_free(ctx)
```

---

ggml_custom_inplace *In-Place Custom Operation Node*

Description

Creates a custom-op node that writes into a view of `a` rather than into a freshly allocated tensor. The output has the shape and type of `a`, and `a` itself is the node's first source; tensors in `args` follow it, so the kernel sees `a` as `dst->src[0]` and the first extra argument as `dst->src[1]`.

Usage

```
ggml_custom_inplace(ctx, a, name, args = NULL, n_tasks = GGML_N_TASKS_MAX)
```

Arguments

`ctx`	GGML context
`a`	Tensor to write into
`name`	Name of a registered custom op (character scalar)
`args`	List of additional input tensors, or NULL for none. At most 8 tensors, one fewer than `ggml_custom` because `a` occupies the first source slot.
`n_tasks`	Number of threads the kernel may be split across, or `GGML_N_TASKS_MAX` (-1). Use 1 for a kernel that is not thread-safe.

Details

See ggml_custom for how kernels are registered and for the CPU-only backend restriction, which applies here too.

Value

Tensor pointer for the custom node (a view of a)

See Also

ggml_custom, ggml_custom_ops

Examples

```
# "clip_inplace" takes its bounds as a 2-element tensor, since a custom node
# carries no scalar parameters of its own.
ggml_set_n_threads(1L)
ctx <- ggml_init(16 * 1024 * 1024)
x <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(x, c(-3, -1, 0, 1, 3))

bounds <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 2)
ggml_set_f32(bounds, c(-1, 1))

y <- ggml_custom_inplace(ctx, x, "clip_inplace", args = list(bounds))
graph <- ggml_build_forward_expand(ctx, y)
ggml_graph_compute(ctx, graph)
ggml_get_f32(y)   # -1, -1, 0, 1, 1
ggml_free(ctx)
```

---

ggml_custom_ops *List Registered Custom Ops*

Description

Returns the names of the C kernels currently registered with ggmlR, in registration order. Useful for checking that a downstream package registered its kernels as expected.

Usage

```
ggml_custom_ops()
```

Value

Character vector of registered custom op names, empty if none

Built-in kernels

ggmlR registers three kernels at load time. Each one exercises a different path through the custom-op API, and they double as worked examples:

`"row_median"` Median of each row. One input; the output shape is chosen by the caller as `ne[1] = 1`, collapsing the row. Use `ne = c(1, nrow)` for an `n x nrow` input.

`"row_permute"` `out[i] = x[perm[i]]` within each row, with `perm` 0-based. Two inputs: F32 data and an I32 index tensor. Indices outside the row yield 0. This is not `ggml_get_rows`, which permutes whole rows rather than elements inside one.

`"clip_inplace"` Clamps values into `[lo, hi]`, writing into the input. For use with `ggml_custom_inplace`; the bounds are passed as a 2-element F32 tensor, since a custom node carries no scalar parameters of its own. `ggml_clamp` does the same thing as a graph op – this kernel exists as the in-place example.

See Also

`ggml_custom`, `ggml_custom_inplace`

Examples

```
ggml_custom_ops()
```

---

ggml_cycles *Get CPU Cycles*

Description

Returns the current CPU cycle count. Useful for low-level benchmarking.

Usage

```
ggml_cycles()
```

Value

Numeric value representing CPU cycles

Examples

```
ggml_cycles()
```

ggml_cycles_per_ms *Get CPU Cycles per Millisecond*

Description

Returns an estimate of CPU cycles per millisecond. Useful for converting cycle counts to time.

Usage

```
ggml_cycles_per_ms()
```

Value

Numeric value representing cycles per millisecond

Examples

```
ggml_cycles_per_ms()
```

ggml_default_mlp *Default MLP builder for classification and regression*

Description

Constructs an uncompiled sequential multi-layer perceptron suitable as a starting point for tabular classification or regression. This is the default `model_fn` used by `LearnerClassifGGML` and `LearnerRegrGGML` when the user does not supply a custom builder, and it is also exported for direct use or as a template for user-defined builders.

Usage

```
ggml_default_mlp(
  n_features,
  n_out,
  task_type = c("classif", "regr"),
  hidden_layers = c(128L, 64L),
  activation = "relu",
  dropout = 0.2
)
```

Arguments

n_features	Integer. Number of input features. Required.
n_out	Integer. Number of output units. For classification this is the number of classes; for regression this is typically 1.
task_type	Character. One of "classif" or "regr". Controls the final layer's activation.
hidden_layers	Integer vector. Widths of the hidden dense layers. Default c(128L, 64L). Pass integer(0) for a linear model.
activation	Character. Activation applied to each hidden layer. Default "relu". Passed through to ggml_layer_dense.
dropout	Numeric in [0, 1). Dropout rate applied after each hidden layer. Set to 0 to disable dropout. Default 0.2.

Details

The returned model is **not compiled**: the caller is responsible for calling ggml_compile with the appropriate loss ("categorical_crossentropy" for classification, "mse" for regression) before training.

The final layer is chosen based on task_type:

- "classif" — dense with units = n_out and softmax activation.
- "regr" — dense with units = n_out and no activation (identity / linear output).

Value

An uncompiled ggml_sequential_model object. Call ggml_compile before ggml_fit.

See Also

ggml_model_sequential, ggml_layer_dense, ggml_layer_dropout, ggml_compile

Examples

```
## Not run:
# 3-class classifier on 20 features
model <- ggml_default_mlp(
  n_features   = 20L,
  n_out        = 3L,
  task_type    = "classif",
  hidden_layers = c(64L, 32L),
  dropout      = 0.1
)
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")

# Single-output regressor
reg <- ggml_default_mlp(
  n_features = 10L,
  n_out      = 1L,
```

```
  task_type  = "regr"
)
reg <- ggml_compile(reg, optimizer = "adam", loss = "mse")

## End(Not run)
```

---

ggml_dense *Create a Dense Layer Object*

Description

Returns a reusable layer object for use with `ggml_apply()`. Applying the same object to multiple tensor nodes shares weights.

Usage

```
ggml_dense(
  units,
  activation = NULL,
  name = NULL,
  trainable = TRUE,
  time_distributed = FALSE
)
```

Arguments

`units`	Number of output units.
`activation`	Activation function name or NULL.
`name`	Optional character name.
`trainable`	Logical; whether weights are updated during training.
`time_distributed`	Logical; apply the kernel independently at every position of a sequence input rather than flattening it (default `FALSE`). A `c(seq_len, features)` parent then yields `c(seq_len, units)` instead of a flat `units`, which is what a transformer's position-wise feed-forward sublayer needs. See `ggml_layer_dense` for details.

Value

A `ggml_layer` object.

Examples

```
encoder <- ggml_dense(64L, activation = "relu")
x1 <- ggml_input(shape = 32L)
x2 <- ggml_input(shape = 32L)
out1 <- x1 |> ggml_apply(encoder)
out2 <- x2 |> ggml_apply(encoder)  # shared weights
```

ggml_diag *Diagonal Matrix (Graph)*

Description

Creates a diagonal matrix from a vector. For vector a[n], produces matrix with a on the diagonal.

Usage

```
ggml_diag(ctx, a)
```

Arguments

ctx	GGML context
a	Input vector tensor

Value

Diagonal matrix tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 3)
ggml_set_f32(a, c(1, 2, 3))
d <- ggml_diag(ctx, a)  # 3x3 diagonal matrix
graph <- ggml_build_forward_expand(ctx, d)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_diag_mask_inf *Diagonal Mask with -Inf (Graph)*

Description

Creates a graph node that sets elements above the diagonal to -Inf. This is used for causal (autoregressive) attention masking.

Usage

```
ggml_diag_mask_inf(ctx, a, n_past)
```

Arguments

ctx	GGML context
a	Input tensor (typically attention scores)
n_past	Number of past tokens (shifts the diagonal). Use 0 for standard causal masking where position i can only attend to positions <= i.

Details

In causal attention, we want each position to only attend to itself and previous positions. Setting future positions to -Inf ensures that after softmax, they contribute 0 attention weight.

The n_past parameter allows for KV-cache scenarios where the diagonal needs to be shifted to account for previously processed tokens.

Value

Tensor with same shape as input, elements above diagonal set to -Inf

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Create attention scores matrix
scores <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 4)
ggml_set_f32(scores, rep(1, 16))
# Apply causal mask
masked <- ggml_diag_mask_inf(ctx, scores, 0)
graph <- ggml_build_forward_expand(ctx, masked)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_diag_mask_inf_inplace

Diagonal Mask with -Inf In-place (Graph)

Description

In-place version of ggml_diag_mask_inf. Returns a view of the input tensor.

Usage

```
ggml_diag_mask_inf_inplace(ctx, a, n_past)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)
n_past	Number of past tokens

Value

View of input tensor with elements above diagonal set to -Inf

---

ggml_diag_mask_zero *Diagonal Mask with Zero (Graph)*

Description

Creates a graph node that sets elements above the diagonal to 0. Alternative to -Inf masking for certain use cases.

Usage

```
ggml_diag_mask_zero(ctx, a, n_past)
```

Arguments

ctx	GGML context
a	Input tensor
n_past	Number of past tokens

Value

Tensor with same shape as input, elements above diagonal set to 0

---

ggml_div *Element-wise Division (Graph)*

Description

Creates a graph node for element-wise division.

Usage

```
ggml_div(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor (numerator)
b	Second tensor (denominator, same shape as a)

Value

Tensor representing the division operation (a / b)

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(10, 20, 30, 40, 50))
ggml_set_f32(b, c(2, 2, 2, 2, 2))
result <- ggml_div(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

---

ggml_div_inplace *Element-wise Division In-place (Graph)*

Description

Creates a graph node for in-place element-wise division. Result is stored in tensor a, saving memory allocation.

Usage

```
ggml_div_inplace(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor (will be modified in-place)
b	Second tensor (same shape as a)

Value

View of tensor a with the division result

---

ggml_dup *Duplicate Tensor (Graph)*

Description

Creates a graph node that copies a tensor. This is a graph operation that must be computed using ggml_build_forward_expand() and ggml_graph_compute(). Unlike ggml_dup_tensor which just allocates, this creates a copy operation in the graph.

Usage

```
ggml_dup(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the copy operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
b <- ggml_dup(ctx, a)
graph <- ggml_build_forward_expand(ctx, b)
ggml_graph_compute(ctx, graph)
ggml_get_f32(b)
ggml_free(ctx)
```

---

ggml_dup_inplace *Duplicate Tensor In-place (Graph)*

Description

Creates a graph node for in-place tensor duplication. Returns a view of the input tensor.

Usage

```
ggml_dup_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

View of tensor a

ggml_dup_tensor *Duplicate Tensor*

Description

Creates a copy of a tensor with the same shape and type

Usage

```
ggml_dup_tensor(ctx, tensor)
```

Arguments

ctx	GGML context
tensor	Tensor to duplicate

Value

New tensor pointer with same shape

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
b <- ggml_dup_tensor(ctx, a)
ggml_nelements(b)
ggml_free(ctx)
```

ggml_element_size *Get Element Size*

Description

Returns the size of a single element in the tensor.

Usage

```
ggml_element_size(tensor)
```

Arguments

tensor	Tensor pointer

Value

Element size in bytes

ggml_elu *ELU Activation (Graph)*

Description

Creates a graph node for ELU (Exponential Linear Unit) activation. ELU(x) = x if x > 0, else alpha * (exp(x) - 1) where alpha = 1.

Usage

```
ggml_elu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the ELU operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
r <- ggml_elu(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)
ggml_free(ctx)
```

ggml_elu_inplace *ELU Activation In-place (Graph)*

Description

Creates a graph node for in-place ELU (Exponential Linear Unit) activation.

Usage

```
ggml_elu_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with ELU applied

---

ggml_embedding *Create an Embedding Layer Object*

Description

Create an Embedding Layer Object

Usage

```
ggml_embedding(vocab_size, dim, name = NULL, trainable = TRUE)
```

Arguments

vocab_size	Number of distinct tokens.
dim	Embedding dimension.
name	Optional character name.
trainable	Logical.

Value

A ggml_layer object.

---

ggml_estimate_memory *Estimate Required Memory*

Description

Helper function to estimate memory needed for a tensor

Usage

```
ggml_estimate_memory(type = GGML_TYPE_F32, ne0, ne1 = 1, ne2 = 1, ne3 = 1)
```

Arguments

type	Tensor type (GGML_TYPE_F32, etc)
ne0	Size of dimension 0
ne1	Size of dimension 1 (optional)
ne2	Size of dimension 2 (optional)
ne3	Size of dimension 3 (optional)

Value

Estimated memory in bytes

Examples

```
# For 1000x1000 F32 matrix
ggml_estimate_memory(GGML_TYPE_F32, 1000, 1000)
```

---

ggml_evaluate.ggml_functional_model

Evaluate a Trained Model

Description

Evaluate a Trained Model

Usage

```
## S3 method for class 'ggml_functional_model'
ggml_evaluate(model, x, y, batch_size = 32L, ...)

ggml_evaluate(model, ...)

## S3 method for class 'ggml_sequential_model'
ggml_evaluate(
  model,
  x,
  y,
  batch_size = 32,
  sample_weight = NULL,
  class_weight = NULL,
  ...
)
```

Arguments

model	A trained ggml_sequential_model
x	Test data
y	Test labels (one-hot encoded)
batch_size	Batch size for evaluation
...	Additional arguments (ignored).
sample_weight	Numeric vector of per-sample weights (length = nrow(x)).
class_weight	Named vector of weights per class, e.g. c("0"=1, "1"=10). Cannot be used with sample_weight.

Value

Named list with `loss` and `accuracy`. A multi-output functional model additionally reports each head as `<output>_loss`. Note the deliberate difference from the training history, which names the same quantities `train_<output>_loss` and `val_<output>_loss`: there the prefix distinguishes the two phases (and keeps an output named `val_x` from colliding with another output's validation column), whereas evaluation has no phases to tell apart, so the bare name is used.

Examples

```
ggml_set_n_threads(1L)  # deterministic, single OpenMP pool
n <- 128
x <- matrix(runif(n * 4), nrow = n, ncol = 4)
y <- matrix(0, nrow = n, ncol = 2)
for (i in seq_len(n)) { y[i, if (sum(x[i,]) > 2) 1L else 2L] <- 1 }

model <- ggml_model_sequential() |>
  ggml_layer_dense(8, activation = "relu") |>
  ggml_layer_dense(2, activation = "softmax")
model$input_shape <- 4L
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")
model <- ggml_fit(model, x, y, epochs = 5, batch_size = 32, verbose = 0)

# Basic evaluation
result <- ggml_evaluate(model, x, y, batch_size = 32)

# With sample_weight
sw <- runif(n, 0.5, 1.5)
result <- ggml_evaluate(model, x, y, batch_size = 32, sample_weight = sw)

# With class_weight
result <- ggml_evaluate(model, x, y, batch_size = 32,
                        class_weight = c("0" = 1, "1" = 2))
```

ggml_exp *Exponential (Graph)*

Description

Creates a graph node for element-wise exponential: exp(x)

Usage

```
ggml_exp(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the exp operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 3)
ggml_set_f32(a, c(0, 1, 2))
result <- ggml_exp(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [1, e, e^2]
ggml_free(ctx)
```

ggml_exp_inplace *Exponential In-place (Graph)*

Description

Creates a graph node for in-place element-wise exponential: e^x

Usage

```
ggml_exp_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with exponential values

`ggml_extract` *Extract a feature-by-cell matrix from a single-cell container*

Description

Pulls an expression matrix out of a Seurat object, a `SingleCellExperiment`, a sparse `dgCMatrix` or a plain `matrix`, returning a dense numeric matrix with features in rows and cells in columns — the layout the GPU engine expects. Optional `genes`/`cells` subsetting happens before any sparse-to-dense materialisation.

Usage

```
ggml_extract(x, assay = NULL, layer = "data", genes = NULL, cells = NULL, ...)

## S3 method for class 'matrix'
ggml_extract(x, assay = NULL, layer = "data", genes = NULL, cells = NULL, ...)

## S3 method for class 'dgCMatrix'
ggml_extract(
  x,
  assay = NULL,
  layer = "data",
  genes = NULL,
  cells = NULL,
  keep_sparse = FALSE,
  ...
)

## S3 method for class 'Seurat'
ggml_extract(
  x,
  assay = NULL,
  layer = "data",
  genes = NULL,
  cells = NULL,
  keep_sparse = FALSE,
  ...
)

## S3 method for class 'SingleCellExperiment'
ggml_extract(
  x,
  assay = NULL,
```

```
  layer = "logcounts",
  genes = NULL,
  cells = NULL,
  keep_sparse = FALSE,
  ...
)
```

Arguments

x	A Seurat, SingleCellExperiment, dgCMatrix or matrix object.
assay	Assay to read from. Seurat: defaults to the object's default assay. Ignored for bare matrices.
layer	Layer / slot to read. Seurat v5: a layer name (default "data"); Seurat v4: mapped to the slot argument of GetAssayData. Ignored for bare matrices.
genes	Optional character/integer vector selecting feature rows.
cells	Optional character/integer vector selecting cell columns.
...	Passed to methods.
keep_sparse	If TRUE, return the (subset) dgCMatrix without densifying. Only the LogNormalize path (op = "normalize") sets this, since it transforms the stored nonzeros in place; every other op leaves it FALSE and gets a dense matrix as before.

Details

For a SingleCellExperiment the layer argument names an assay (default "logcounts", the log-normalised matrix); it is read with SummarizedExperiment::assay().

Value

A dense numeric matrix, features x cells.

---

ggml_fit.ggml_functional_model

Train a Model (dispatcher)

Description

Dispatcher: if the first argument is a ggml_sequential_model, delegates to the Keras-style high-level API (ggml_fit_sequential); otherwise delegates to the low-level optimizer loop (ggml_fit_opt).

Usage

```
## S3 method for class 'ggml_functional_model'
ggml_fit(
  model,
  x,
  y,
```

```
  epochs = 1L,
  batch_size = 32L,
  validation_split = 0,
  validation_data = NULL,
  verbose = 1L,
  shuffle = TRUE,
  callbacks = list(),
  ...
)

ggml_fit(model, ...)

## S3 method for class 'ggml_sequential_model'
ggml_fit(model, ...)

## Default S3 method:
ggml_fit(model, ...)
```

Arguments

model	A compiled model object.
x	Training data (matrix or array).
y	Training labels (matrix, one-hot encoded).
epochs	Number of training epochs (default: 1).
batch_size	Batch size (default: 32).
validation_split	Fraction of data for validation (default: 0).
validation_data	Optional list(x_val, y_val). Overrides validation_split.
verbose	0 = silent, 1 = progress (default: 1).
shuffle	Shuffle the data (default TRUE). The dataset is shuffled once before the train/validation split, then the training portion is reshuffled each epoch while the validation portion stays fixed. Set to FALSE for time series or exactly reproducible runs.
callbacks	List of callback objects, e.g. ggml_callback_early_stopping or an LR scheduler. Each is a list with on_epoch_begin(epoch, logs, state) and/or on_epoch_end(epoch, logs, state); setting state$stop <- TRUE stops training. logs holds train_loss, train_accuracy, val_loss and val_accuracy, and for a multi-output model also train_<output>_loss, train_<output>_accuracy, val_<output>_loss and val_<output>_accuracy, so a callback can monitor one head rather than the total. These are the same names model$history uses, so a monitor= naming one of them matches a column there. Both phases carry a prefix on purpose: it keeps an output named val_x from colliding with another output's validation key. ggml_evaluate reports the same per-head quantities under the bare <output>_loss, since a single evaluation has no train/val phases to tell apart.
...	Arguments passed to the appropriate implementation.

Details

Keras-style (Sequential model):

model A compiled `ggml_sequential_model`

x Training data (matrix or array)

y Training labels (matrix, one-hot encoded for classification)

epochs Number of training epochs (default: 1)

batch_size Batch size (default: 32)

validation_split Fraction of data for validation (default: 0)

validation_data Optional list(x_val, y_val) for validation. Overrides validation_split.

class_weight Named vector of weights per class, e.g. c("0"=1, "1"=10). Cannot be used with sample_weight.

sample_weight Per-sample weights: a numeric vector of length `nrow(x)`, or – for `mean_squared_error` only – a `nrow(x)` by `ncol(y)` matrix acting as a per-output loss mask. A mask trains only the outputs whose weight is non-zero, which is what a Q-model needs: 1 on the action that has a TD target, 0 elsewhere. The denominator is unchanged (`sum(w*(pred-y)^2)/nelements`), so masking does not renormalise over the active outputs. Cannot be used with class_weight.

verbose 0 = silent, 1 = progress (default: 1)

shuffle Shuffle the data (default: TRUE). Shuffled once before the train/validation split, then the training portion each epoch; the validation portion stays fixed. FALSE for time series or exactly reproducible runs.

callbacks List of callback objects (early stopping, LR schedules)

generator Function of no arguments yielding `list(x, y)` for one batch, or `NULL` when exhausted – an alternative to `x` and `y` that keeps only the current batch in memory. See `ggml_fit_generator`, to which the call is forwarded along with `steps_per_epoch`, `initial_epoch`, `validation_generator` and `validation_steps`. The arguments that are defined in terms of a whole dataset – `validation_split`, `validation_data`, `class_weight` and `sample_weight` – are rejected rather than ignored, and `shuffle` is inert because the generator owns its own order.

Low-level (optimizer loop):

sched Backend scheduler

ctx_compute Compute context

inputs Input tensor

outputs Output tensor

dataset Dataset from `ggml_opt_dataset_init()`

loss_type Loss type (default: MSE)

optimizer Optimizer type (default: AdamW)

nepoch Number of epochs (default: 10)

nbatch_logical Logical batch size (default: 32)

val_split Validation fraction (default: 0)

shuffle Shuffle the data (default: TRUE)

callbacks List of callback objects

silent Suppress output (default: FALSE)

Value

For Sequential models: the trained model (invisibly). For the low-level API: a data frame with columns epoch, `train_loss`, `train_accuracy`, `val_loss`, `val_accuracy`.

See Also

ggml_fit_opt, ggml_compile

Examples

```
ggml_set_n_threads(1L)  # deterministic, single OpenMP pool
n <- 128
x <- matrix(runif(n * 4), nrow = n, ncol = 4)
y <- matrix(0, nrow = n, ncol = 2)
for (i in seq_len(n)) { y[i, if (sum(x[i,]) > 2) 1L else 2L] <- 1 }

model <- ggml_model_sequential() |>
  ggml_layer_dense(8, activation = "relu") |>
  ggml_layer_dense(2, activation = "softmax")
model$input_shape <- 4L
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")

# Basic training
model <- ggml_fit(model, x, y, epochs = 5, batch_size = 32, verbose = 0)

# With validation_data
x_val <- matrix(runif(32 * 4), nrow = 32, ncol = 4)
y_val <- matrix(0, nrow = 32, ncol = 2)
for (i in seq_len(32)) { y_val[i, if (sum(x_val[i,]) > 2) 1L else 2L] <- 1 }
model <- ggml_fit(model, x, y, epochs = 3, batch_size = 32,
                  validation_data = list(x_val, y_val), verbose = 0)

# With class_weight (useful for imbalanced classes)
model <- ggml_fit(model, x, y, epochs = 3, batch_size = 32,
                  class_weight = c("0" = 1, "1" = 2), verbose = 0)

# With sample_weight
sw <- runif(n, 0.5, 1.5)
model <- ggml_fit(model, x, y, epochs = 3, batch_size = 32,
                  sample_weight = sw, verbose = 0)
```

---

ggml_fit_generator *Train a Sequential Model from a Generator*

Description

Streaming counterpart of ggml_fit for sequential models: batches are pulled from a generator instead of being sliced out of a matrix held in memory. Reachable either directly or as `ggml_fit(model, generator = ...)`.

Usage

```
ggml_fit_generator(
  model,
  generator,
  steps_per_epoch = NULL,
  epochs = 1L,
  batch_size = 32L,
  initial_epoch = 0L,
  validation_generator = NULL,
  validation_steps = NULL,
  calibration_generator = NULL,
  calibration_steps = NULL,
  verbose = 1,
  callbacks = list()
)
```

Arguments

model	A compiled ggml_sequential_model
generator	Function of no arguments yielding list(x, y) for one batch, or NULL to end the epoch. Mutually exclusive with x/ y. Batches are in the same R layout those arguments take, and must contain exactly batch_size samples. A finite generator has to reset its own state before returning NULL, so that the next epoch starts over; see ggml_fit_opt_gen for the full contract and an example.
steps_per_epoch	Batches per epoch. Required for a generator that never returns NULL; otherwise the epoch ends when it does.
epochs	Number of epochs to run
batch_size	Samples per batch. The graph is built for a fixed batch, so the generator must yield exactly this many; a short batch is an error rather than being padded.
initial_epoch	Epoch number to start counting from. Shifts the numbering seen by callbacks and learning-rate schedules; it does not restore optimizer state (see ggml_fit_opt_gen).
validation_generator	Optional generator for validation, same contract.
validation_steps	Validation batches per epoch.
calibration_generator	Generator used to recompute batch-norm running statistics after training. Defaults to generator; note that a finite generator already exhausted by training yields nothing, so pass a fresh one.
calibration_steps	Batches to draw for that calibration.
verbose	0 to silence per-epoch messages
callbacks	List of callback lists, as for ggml_fit

Details

Batches are given in the same R layout that `x` and `y` use; the column-major conversion ggml wants happens internally, which is also what lets a recorded batch feed batch-norm calibration unchanged.

Value

The trained model, invisibly.

See Also

`ggml_fit_opt_gen` for the low-level loop this wraps.

`ggml_fit_opt` *Fit model with R-side epoch loop and callbacks*

Description

Trains a model epoch by epoch in R, allowing callbacks for early stopping and learning rate scheduling. Optimizer state (momentum) is preserved across all epochs.

Usage

```
ggml_fit_opt(
  sched,
  ctx_compute,
  inputs,
  outputs,
  dataset,
  loss_type = ggml_opt_loss_type_mse(),
  optimizer = ggml_opt_optimizer_type_adamw(),
  nepoch = 10L,
  nbatch_logical = 32L,
  val_split = 0,
  shuffle = TRUE,
  shuffle_all = shuffle,
  callbacks = list(),
  silent = FALSE,
  loss_mask_ne0 = NULL,
  weight_decay = 0
)
```

Arguments

`sched`	Backend scheduler
`ctx_compute`	Compute context (for temporary tensors)
`inputs`	Input tensor with shape [ne_datapoint, batch_size]

outputs	Output tensor with shape [ne_label, batch_size]
dataset	Dataset created with ‘ggml_opt_dataset_init()‘
loss_type	Loss type (default: MSE)
optimizer	Optimizer type (default: AdamW)
nepoch	Number of epochs
nbatch_logical	Logical batch size (for gradient accumulation)
val_split	Fraction of data for validation (0.0 to 1.0)
shuffle	Shuffle the training portion each epoch (default TRUE). Set to FALSE for time series or to make a run exactly reproducible.
shuffle_all	Shuffle the whole dataset once before the train/validation split is taken, so the validation portion is a random sample rather than the tail of the input. Defaults to shuffle. Callers that append an explicit validation set to the data pass FALSE, since there the split is positional and must be preserved.
callbacks	List of callback lists. Each element may have ‘on_epoch_begin(epoch, logs, state)‘ and/or ‘on_epoch_end(epoch, logs, state)‘. Built-in factories: ‘ggml_callback_early_stopping()‘, ‘ggml_schedule_step_decay()‘, ‘ggml_schedule_cosine_decay()‘, ‘ggml_schedule_reduce_on_plateau()‘ ‘state‘ is a mutable environment with fields: ‘stop‘ (set TRUE to stop training), ‘lr_ud‘, ‘nepoch‘.
silent	Whether to suppress per-epoch progress output
loss_mask_ne0	Width of the weighted-MSE weight tensor: NULL or 1 for per-datapoint weights, or the output width for a per-output mask. Ignored by every other loss type.
weight_decay	Decoupled weight decay in [0, 1], as for ggml_opt_init_for_fit; 0 turns it off.

Value

Data frame with columns epoch, train_loss, train_accuracy, val_loss, val_accuracy

See Also

Other optimization: ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_
ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_ba
ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(
ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(),
ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(),
ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(),
ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(),
ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(),
ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh
ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(),
ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(),
ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(),
ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(),
ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(),
ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

Examples

```
if (FALSE) {
history <- ggml_fit_opt(sched, ctx_compute, inputs, outputs, dataset,
  nepoch = 50, val_split = 0.2,
  callbacks = list(
    ggml_callback_early_stopping(monitor = "val_loss", patience = 5),
    ggml_schedule_cosine_decay()
  ))
}
```

---

ggml_fit_opt_gen *Train from a generator, one batch at a time*

Description

Streaming counterpart of ggml_fit_opt. Instead of a ggml_opt_dataset holding every sample, batches are pulled from a generator function, so only the current batch is ever in memory. This is what makes three things possible that the dataset path cannot express: data larger than RAM, augmentation that must differ per epoch, and generated sources (RL rollouts, simulators) that have no fixed dataset at all.

Usage

```
ggml_fit_opt_gen(
  sched,
  ctx_compute,
  inputs,
  outputs,
  generator,
  steps_per_epoch = NULL,
  epochs = 1L,
  initial_epoch = 0L,
  validation_generator = NULL,
  validation_steps = NULL,
  loss_type = ggml_opt_loss_type_mse(),
  optimizer = ggml_opt_optimizer_type_adamw(),
  nbatch_logical = NULL,
  callbacks = list(),
  silent = FALSE,
  loss_mask_ne0 = NULL,
  weight_decay = 0
)
```

Arguments

sched	Backend scheduler
ctx_compute	Compute context holding the model graph

inputs	Input tensor of the built graph
outputs	Output tensor of the built graph
generator	Function of no arguments returning one batch, or NULL when exhausted. See the contract above.
steps_per_epoch	Number of batches that make up one epoch. Required when generator is infinite; when omitted, an epoch runs until the generator returns NULL.
epochs	Number of epochs to run.
initial_epoch	Epoch number to start counting from, as in Keras. This shifts the numbering seen by callbacks and learning-rate schedules only; it does not restore optimizer state. A fresh opt_ctx starts with zeroed Adam moments regardless, so this is a warm restart with the LR schedule resumed, not a full resume.
validation_generator	Optional generator used for validation after each epoch, following the same contract. Evaluated without a backward pass.
validation_steps	Number of validation batches per epoch. Required when validation_generator is infinite.
loss_type	Loss type constant
optimizer	Optimizer type constant
nbatch_logical	Logical batch size, for gradient accumulation
callbacks	List of callback lists, as for ggml_fit_opt
silent	Suppress per-epoch messages
loss_mask_ne0	Width of the per-output loss mask, or NULL
weight_decay	Decoupled weight decay in [0, 1], as for ggml_opt_init_for_fit; 0 turns it off.

Value

A history data frame, one row per epoch, as ggml_fit_opt returns.

The generator contract

generator is a function of no arguments. Each call returns one batch; returning NULL ends the epoch. The same contract covers both finite and infinite sources – an infinite generator simply never returns NULL, and steps_per_epoch then becomes required, because nothing else can say where an epoch ends.

A finite generator must restart itself. Training runs several epochs over the same source, so a generator that ends must reset its own state before returning NULL, leaving the next call to yield the first batch again. Nothing else can do it: a bare closure has no handle to reset from. An epoch that receives no batch at all stops training with a warning rather than looping over nothing.

```
batch_gen <- function(x, y, batch_size) {
  i <- 0L
  nb <- nrow(x) %/% batch_size
```

```
  function() {
    if (i >= nb) {
      i <<- 0L          # reset, so the next epoch starts over
      return(NULL)
    }
    idx <- (i * batch_size + 1L):((i + 1L) * batch_size)
    i <<- i + 1L
    list(x[idx, , drop = FALSE], y[idx, , drop = FALSE])
  }
}
```

A batch is `list(inputs = list(...), labels = list(...))`. For the common single-input case a plain `list(x, y)` is accepted as well. Each element is a numeric vector or array already in ggml layout, holding exactly one physical batch.

The generator is called from R, so it may do anything R can: read a shard from disk, augment the previous batch, or sample a replay buffer that a simulator is filling.

Gradient accumulation

When `nbatch_logical` exceeds the graph's physical batch size, the optimizer accumulates gradients over several generator batches and steps once per logical batch. The period is tracked inside `opt_ctx`; the generator is unaware of it and simply keeps producing physical batches.

See Also

`ggml_fit_opt` for the in-memory dataset path.

Other optimization: `ggml_fit_opt()`, `ggml_fit_opt_multi()`, `ggml_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `ggml_opt_dataset_get_batch_host()`, `ggml_opt_dataset_init()`, `ggml_opt_dataset_labels()`, `ggml_opt_dataset_ndata()`, `ggml_opt_dataset_shuffle()`, `ggml_opt_dataset_weights()`, `ggml_opt_default_params()`, `ggml_opt_epoch()`, `ggml_opt_eval()`, `ggml_opt_fit()`, `ggml_opt_fit_multi()`, `ggml_opt_free()`, `ggml_opt_get_lr()`, `ggml_opt_grad_acc()`, `ggml_opt_init()`, `ggml_opt_init_for_fit()`, `ggml_opt_init_for_fit_multi()`, `ggml_opt_inputs()`, `ggml_opt_labels()`, `ggml_opt_loss()`, `ggml_opt_loss_i()`, `ggml_opt_loss_type_binary_cross_entropy()`, `ggml_opt_loss_type_cross_entropy()`, `ggml_opt_loss_type_huber()`, `ggml_opt_loss_type_mae()`, `ggml_opt_loss_type_mean()`, `ggml_opt_loss_type_mse()`, `ggml_opt_loss_type_sum()`, `ggml_opt_loss_type_weighted()`, `ggml_opt_n_loss()`, `ggml_opt_ncorrect()`, `ggml_opt_optimizer_name()`, `ggml_opt_optimizer_type_adamw()`, `ggml_opt_optimizer_type_sgd()`, `ggml_opt_outputs()`, `ggml_opt_outputs_i()`, `ggml_opt_pred()`, `ggml_opt_prepare_alloc()`, `ggml_opt_reset()`, `ggml_opt_result_accuracy()`, `ggml_opt_result_free()`, `ggml_opt_result_init()`, `ggml_opt_result_loss()`, `ggml_opt_result_loss_i()`, `ggml_opt_result_n_loss()`, `ggml_opt_result_ndata()`, `ggml_opt_result_pred()`, `ggml_opt_result_reset()`, `ggml_opt_set_lr()`, `ggml_opt_static_graphs()`, `ggml_snapshot_free()`, `ggml_trainer()`

`ggml_fit_opt_multi` *Fit a multi-output model with an R-side epoch loop and callbacks*

Description

Multi-output counterpart of `ggml_fit_opt()`. Each output head has its own labels, loss type and weight; the optimized total is `sum(loss_weights[i] * loss_i)`. Because the epoch loop runs in R, this path supports callbacks (early stopping, LR schedulers) and the `shuffle`/`shuffle_all` controls, which the single C call `ggml_opt_fit_multi()` cannot offer.

Usage

```
ggml_fit_opt_multi(
  sched,
  ctx_compute,
  inputs,
  outputs,
  dataset,
  loss_types,
  loss_weights = rep(1, length(outputs)),
  labels_offs = NULL,
  head_names = NULL,
  optimizer = ggml_opt_optimizer_type_adamw(),
  nepoch = 10L,
  nbatch_logical = 32L,
  val_split = 0,
  shuffle = TRUE,
  shuffle_all = shuffle,
  callbacks = list(),
  silent = FALSE,
  weight_decay = 0
)
```

Arguments

`sched`	Backend scheduler
`ctx_compute`	Compute context holding the model graph
`inputs`	Input tensor with shape [ne_datapoint, batch_size]
`outputs`	List of output tensors, one per head
`dataset`	Dataset with data and concatenated labels
`loss_types`	Integer vector of loss type constants, one per head
`loss_weights`	Numeric vector of head weights (default: all 1)
`labels_offs`	Numeric vector of 0-based label offsets, one per head. Defaults to laying the heads out back to back in head order, using each head's `ne[0]` as its width.
`head_names`	Character vector naming the heads, one per head, used for the per-head keys in `logs` and as the column names of the returned matrices. Defaults to `head_1`, `head_2`, ... Callers that have real output names should pass them, so a callback's `monitor=` matches what the training history shows.
`optimizer`	Optimizer type (default: AdamW)

nepoch	Number of epochs
nbatch_logical	Logical batch size (for gradient accumulation)
val_split	Fraction of data for validation (0.0 to 1.0)
shuffle	Shuffle the training portion each epoch (default TRUE). Set to FALSE for time series or to make a run exactly reproducible.
shuffle_all	Shuffle the whole dataset once before the train/validation split is taken, so the validation portion is a random sample rather than the tail of the input. Defaults to shuffle.
callbacks	List of callback lists, as for ggml_fit_opt(). Each element may have 'on_epoch_begin(epoch, logs, state)' and/or 'on_epoch_end(epoch, logs, state)'. 'state' is a mutable environment with 'stop', 'lr_ud' and 'nepoch'. 'logs' carries the aggregate metrics ('train_loss', 'train_accuracy', 'val_loss', 'val_accuracy') and, for each head, 'train_<head>_loss', 'train_<head>_accuracy', 'val_<head>_loss' and 'val_<head>_accuracy' – so a scheduler or early stopping can monitor one head rather than the total. Both phases are prefixed, so a head named 'val_x' cannot collide with another head's validation key. Per-head validation keys are 'NA' when 'val_split' is 0, and accuracy keys are 'NA' for heads whose loss is not cross-entropy.
silent	Whether to suppress per-epoch progress output
weight_decay	Decoupled weight decay in [0, 1], as for ggml_opt_init_for_fit; 0 turns it off.

Details

All heads share the dataset's datapoints. The dataset's labels hold every head's labels concatenated along the first dimension: head i occupies labels_offs[i] + seq_len(width_i) of each label row.

Value

Data frame with columns epoch, train_loss, train_accuracy, val_loss, val_accuracy – one row per epoch actually run, so a callback that stops early shortens it. The per-head metrics come as attributes rather than columns, since they are matrices, each [nepoch_run x n_head] with columns named after the heads: attr(h, "head_loss"), attr(h, "head_accuracy"), attr(h, "val_head_loss") and attr(h, "val_head_accuracy"), plus attr(h, "head_names").

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_ba, ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(, ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh, ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(),

ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

Examples

```
if (FALSE) {
history <- ggml_fit_opt_multi(sched, ctx_compute, inputs, list(out_a, out_b),
  dataset, loss_types = c(ggml_opt_loss_type_mse(), ggml_opt_loss_type_mse()),
  nepoch = 50, val_split = 0.2,
  callbacks = list(ggml_callback_early_stopping(monitor = "val_loss", patience = 5)))
attr(history, "head_loss")
}
```

---

ggml_flash_attn_back *Flash Attention Backward (Graph)*

Description

Backward pass for ggml_flash_attn_ext: given the gradient of the attention output, produces the gradients of Q, K and V in one pass.

Usage

```
ggml_flash_attn_back(ctx, q, k, v, mask, d, scale)
```

Arguments

ctx	GGML context
q	Query tensor [DK, N, H, B], as passed to the forward pass
k	Key tensor [DK, M, H_kv, B], as passed to the forward pass
v	Value tensor [DV, M, H_kv, B], as passed to the forward pass
mask	Attention mask [M, N, H_m, B_m] of type F16, or NULL
d	Gradient of the forward output, shape [DV, H, N, B]
scale	Softmax scale used in the forward pass (e.g. 1/sqrt(DK))

Details

Upstream ggml ships this operation as a stub that aborts, left behind when flash_attn was replaced by flash_attn_ext; ggmlR implements it so attention can be trained through the fused op. It is also wired into the graph autodiff, so ggml_build_backward_expand reaches it on its own — calling this directly is only needed when building a backward graph by hand.

The three gradients come back packed head to tail in one contiguous 1D tensor: grad_q, then grad_k, then grad_v, each padded to GGML_MEM_ALIGN. Take a view of the slice you need.

Not supported: max_bias (ALiBi), logit_softcap and attention sinks. Q, K, V and d must be F32.

Value

Tensor holding grad_q, grad_k and grad_v packed contiguously

See Also

ggml_flash_attn_ext for the forward pass.

`ggml_flash_attn_ext` *Flash Attention (Graph)*

Description

Creates a graph node for Flash Attention computation. This is a memory-efficient implementation of scaled dot-product attention.

Usage

```
ggml_flash_attn_ext(
  ctx,
  q,
  k,
  v,
  mask = NULL,
  scale,
  max_bias = 0,
  logit_softcap = 0
)
```

Arguments

`ctx`	GGML context
`q`	Query tensor of shape [head_dim, n_head, n_tokens, batch]
`k`	Key tensor of shape [head_dim, n_head_kv, n_kv, batch]
`v`	Value tensor of shape [head_dim, n_head_kv, n_kv, batch]
`mask`	Optional attention mask tensor (NULL for no mask). For causal attention, use ggml_diag_mask_inf instead.
`scale`	Attention scale factor, typically 1/sqrt(head_dim)
`max_bias`	Maximum ALiBi bias (0.0 to disable ALiBi)
`logit_softcap`	Logit soft-capping value (0.0 to disable). Used by some models like Gemma 2.

Details

Flash Attention computes: softmax(Q * K^T / scale + mask) * V

Key features: - Memory efficient: O(n) instead of O(n^2) memory for attention matrix - Supports grouped-query attention (GQA) when n_head_kv < n_head - Supports multi-query attention (MQA) when n_head_kv = 1 - Optional ALiBi (Attention with Linear Biases) for position encoding - Optional logit soft-capping for numerical stability

Value

Attention output tensor of shape [head_dim, n_head, n_tokens, batch]

Examples

```
ctx <- ggml_init(64 * 1024 * 1024)
head_dim <- 64
n_head <- 8
n_head_kv <- 2  # GQA with 4:1 ratio
seq_len <- 32
q <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, head_dim, n_head, seq_len, 1)
k <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, head_dim, n_head_kv, seq_len, 1)
v <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, head_dim, n_head_kv, seq_len, 1)
ggml_set_f32(q, rnorm(head_dim * n_head * seq_len))
ggml_set_f32(k, rnorm(head_dim * n_head_kv * seq_len))
ggml_set_f32(v, rnorm(head_dim * n_head_kv * seq_len))
# Scale = 1/sqrt(head_dim)
scale <- 1.0 / sqrt(head_dim)
# Compute attention
out <- ggml_flash_attn_ext(ctx, q, k, v, NULL, scale, 0.0, 0.0)
graph <- ggml_build_forward_expand(ctx, out)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_flash_attn_ext_add_sinks

Attach Attention Sinks to a Flash-Attention Node

Description

Adds per-head "sink" logits, an always-available extra key that lets a query attend to nothing in particular.

Usage

```
ggml_flash_attn_ext_add_sinks(tensor, sinks)
```

Arguments

tensor	A flash-attention node
sinks	Sink logits, one per head

Value

The tensor (for chaining)

See Also

ggml_soft_max_add_sinks

ggml_flash_attn_ext_get_prec

Get Flash-Attention Precision

Description

Get Flash-Attention Precision

Usage

```
ggml_flash_attn_ext_get_prec(tensor)
```

Arguments

tensor	A flash-attention node

Value

The precision constant currently set on the node

See Also

ggml_flash_attn_ext_set_prec

ggml_flash_attn_ext_set_prec

Set Flash-Attention Precision

Description

Sets the precision a ggml_flash_attn_ext() node computes in. GGML_PREC_F32 raises accuracy at some cost in speed.

Usage

```
ggml_flash_attn_ext_set_prec(tensor, prec)
```

Arguments

tensor	A flash-attention node
prec	Precision constant

Value

The tensor (for chaining)

See Also

ggml_flash_attn_ext_get_prec

---

ggml_floor *Floor (Graph)*

Description

Creates a graph node for element-wise floor: floor(x)

Usage

```
ggml_floor(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the floor operation

---

ggml_floor_inplace *Floor In-place (Graph)*

Description

Creates a graph node for in-place element-wise floor.

Usage

```
ggml_floor_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with floor values

ggml_free *Free GGML context*

Description

Free GGML context

Usage

```
ggml_free(ctx)
```

Arguments

ctx	Context pointer

Value

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_free(ctx)
```

ggml_freeze_bn_stats *Fix or Release Batch-Norm Statistics*

Description

`ggml_freeze_bn_stats()` fixes the running mean and variance of batch-norm layers: they then normalize with those stored statistics during training as well as at prediction, and `ggml_fit` no longer recalibrates them. `gamma` and `beta` still train unless the layer is also frozen with `ggml_freeze_weights` – the usual fine-tuning setting when batches are small. `ggml_unfreeze_bn_stats()` returns the layers to batch statistics in training and recalibration after a fit, which is the default.

Usage

```
ggml_freeze_bn_stats(
  model,
  from = 1L,
  to = length(model$layers),
  layer_names = NULL,
  prefix = NULL,
  ...
)
```

```
ggml_unfreeze_bn_stats(
  model,
  from = 1L,
  to = length(model$layers),
  layer_names = NULL,
  prefix = NULL,
  ...
)
```

Arguments

model	A model object (ggml_sequential_model or ggml_functional_model)
from	Integer index of the first layer to freeze (default: 1)
to	Integer index of the last layer to freeze (default: last layer)
layer_names	Character vector of layer names to freeze (overrides from/to)
prefix	Character vector: freeze every layer whose name starts with one of these – a whole transformer block, say: prefix = "block1_" reaches block1_norm1, block1_attn, block1_norm2, block1_ff1 and block1_ff2 – not block10_*. Given with layer_names, the selection is the union of the two; every name and every prefix must match at least one layer, or the call stops.
...	Additional arguments passed to methods

Details

Freezing the weights alone does not fix the statistics: a frozen batch-norm still normalizes each training batch by its own mean and variance, and its running statistics are still recalibrated after ggml_fit().

The fixed state is saved by ggml_save_model. Only sequential models carry running statistics; the functional batch_norm is a per-sample RMS normalization (see ggml_layer_batch_norm), already independent of the batch, and is refused with a message.

Value

The model with the selected batch-norm layers changed. Selected layers of other types are skipped; a selection with no batch-norm layer at all is an error.

---

ggml_freeze_weights *Freeze Layer Weights*

Description

Sets trainable = FALSE on layers, preventing their weights from being updated during training. Layers are chosen by layer_names and/or prefix; without either, by the index range from / to (sequential models), or all of them.

Usage

```
ggml_freeze_weights(
  model,
  from = 1L,
  to = length(model$layers),
  layer_names = NULL,
  prefix = NULL,
  ...
)
```

Arguments

model	A model object (ggml_sequential_model or ggml_functional_model)
from	Integer index of the first layer to freeze (default: 1)
to	Integer index of the last layer to freeze (default: last layer)
layer_names	Character vector of layer names to freeze (overrides from/to)
prefix	Character vector: freeze every layer whose name starts with one of these – a whole transformer block, say: prefix = "block1_" reaches block1_norm1, block1_attn, block1_norm2, block1_ff1 and block1_ff2 – not block10_*. Given with layer_names, the selection is the union of the two; every name and every prefix must match at least one layer, or the call stops.
...	Additional arguments passed to methods

Value

The model with selected layers frozen.

See Also

ggml_freeze_bn_stats to fix batch-norm statistics, which freezing the weights does not do.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(64, activation = "relu") |>
  ggml_layer_dense(10, activation = "softmax")

# Freeze all layers
model <- ggml_freeze_weights(model)

# Freeze only the first layer
model <- ggml_freeze_weights(model, from = 1, to = 1)
```

ggml_ftype_to_ggml_type *Convert ftype to ggml_type*

Description

Converts a file type (ftype) to the corresponding GGML type. Used when loading quantized models.

Usage

```
ggml_ftype_to_ggml_type(ftype)
```

Arguments

ftype	File type constant

Value

Integer GGML type

See Also

Other type_system: ggml_blck_size(), ggml_is_quantized(), ggml_type_name(), ggml_type_sizef()

ggml_gallocr_alloc_graph *Allocate Memory for Graph*

Description

Allocates memory for all tensors in the computation graph. This must be called before computing the graph.

Usage

```
ggml_gallocr_alloc_graph(galloc, graph)
```

Arguments

galloc	Graph allocator object
graph	Graph object

Value

TRUE on success, FALSE on failure

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
galloc <- ggml_gallocr_new()

# Create graph
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, b)

# Allocate and compute
ggml_gallocr_alloc_graph(galloc, graph)
ggml_graph_compute(ctx, graph)

ggml_gallocr_free(galloc)
ggml_free(ctx)
```

---

ggml_gallocr_free *Free Graph Allocator*

Description

Frees a graph allocator and all associated buffers.

Usage

```
ggml_gallocr_free(galloc)
```

Arguments

galloc	Graph allocator object

Value

No return value, called for side effects

---

ggml_gallocr_get_buffer_size *Get Graph Allocator Buffer Size*

Description

Returns the size of the buffer used by the graph allocator.

Usage

```
ggml_gallocr_get_buffer_size(galloc, buffer_id = 0L)
```

Arguments

galloc	Graph allocator object
buffer_id	Buffer ID (default: 0 for single-buffer allocator)

Value

Size in bytes

---

ggml_gallocr_new *Create Graph Allocator*

Description

Creates a new graph allocator for efficient memory management. The allocator can automatically allocate and reuse memory for graph tensors.

Usage

```
ggml_gallocr_new()
```

Value

Graph allocator object (external pointer)

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
galloc <- ggml_gallocr_new()

a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, b)

# Allocate graph
ggml_gallocr_alloc_graph(galloc, graph)

ggml_gallocr_free(galloc)
ggml_free(ctx)
```

ggml_gallocr_new_buft *Create a Graph Allocator for a Specific Buffer Type*

Description

Like ggml_gallocr_new, but allocates from the given buffer type rather than the CPU default – so a graph can be allocated in GPU memory.

Usage

```
ggml_gallocr_new_buft(buft)
```

Arguments

buft	Buffer type (e.g. from ggml_backend_get_default_buffer_type())

Value

Graph allocator object (external pointer)

See Also

ggml_gallocr_new, ggml_gallocr_alloc_graph

ggml_gallocr_reserve *Reserve Memory for Graph*

Description

Pre-allocates memory for a graph. This is optional but recommended when running the same graph multiple times to avoid reallocation.

Usage

```
ggml_gallocr_reserve(galloc, graph)
```

Arguments

galloc	Graph allocator object
graph	Graph object

Value

TRUE on success, FALSE on failure

`ggml_gated_linear_attn` *Gated Linear Attention*

Description

Linear attention with a per-token gate (the GLA / RetNet family). Shares the RWKV recurrence machinery: keys, values and queries drive a running state that `g` gates, and `scale` multiplies the query.

Usage

```
ggml_gated_linear_attn(ctx, k, v, q, g, state, scale)
```

Arguments

`ctx`	GGML context
`k`	Keys, `[S, H, n_tokens]`
`v`	Values, `[S, H, n_tokens]`
`q`	Queries, `[S, H, n_tokens]`
`g`	Gate, `[S, H, n_tokens]`
`state`	Initial state, `S * S * H * n_seqs` elements
`scale`	Scalar applied to the queries (usually `1 / sqrt(S)`)

Value

A tensor `[S * H, n_tokens + S * n_seqs]`

Shapes

`k`, `v`, `q` and `g` are `[S, H, n_tokens]`; `state` holds `S * S * H * n_seqs` elements. The result is `[S * H, n_tokens + S * n_seqs]`, outputs then final state – see `ggml_rwkv_output`.

`n_tokens` counts the tokens of *all* sequences together, so it must be a multiple of `n_seqs`.

Backend support

CPU only. Unlike the RWKV and SSM ops, `GGML_OP_GATED_LINEAR_ATTN` has no Vulkan shader, so a graph containing it falls back to the CPU for this node. It has no backward pass either, so it is inference-only.

See Also

`ggml_rwkv_wkv7`, `ggml_rwkv_output`

Other state-space: `ggml_gated_linear_attn_back()`, `ggml_rwkv_output()`, `ggml_rwkv_wkv6()`, `ggml_rwkv_wkv6_back()`, `ggml_rwkv_wkv7()`, `ggml_rwkv_wkv7_back()`, `ggml_ssm_conv()`, `ggml_ssm_conv_back()`, `ggml_ssm_scan()`, `ggml_ssm_scan_back()`, `ggml_ssm_scan_output()`

ggml_gated_linear_attn_back *Backward Pass of Gated Linear Attention*

Description

Gradients of ggml_gated_linear_attn, packed in forward source order: [d_k | d_v | d_q | d_g | d_state].

Usage

```
ggml_gated_linear_attn_back(ctx, k, v, q, g, state, grad, scale)
```

Arguments

ctx	GGML context
k, v, q, g, state	The tensors passed to the forward op
grad	Gradient w.r.t. the packed forward result
scale	The scale the forward op was built with

Value

A flat tensor holding the five gradients, concatenated

See Also

ggml_gated_linear_attn

Other state-space: ggml_gated_linear_attn(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

ggml_geglu *GeGLU (GELU Gated Linear Unit) (Graph)*

Description

Creates a graph node for GeGLU operation. GeGLU uses GELU as the activation function on the first half. CRITICAL for models like GPT-NeoX and Falcon.

Usage

```
ggml_geglu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (first dimension must be even)

Details

Formula: output = GELU(x) * gate

Value

Tensor with half the first dimension of input

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 8, 3)
ggml_set_f32(a, rnorm(24))
r <- ggml_geglu(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # Shape: 4x3
ggml_free(ctx)
```

---

ggml_geglu_quick *GeGLU Quick (Fast GeGLU) (Graph)*

Description

Creates a graph node for fast GeGLU approximation. Uses faster but less accurate GELU approximation for gating.

Usage

```
ggml_geglu_quick(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (first dimension must be even)

Value

Tensor with half the first dimension of input

ggml_geglu_split *GeGLU Split (Graph)*

Description

Creates a graph node for GeGLU with separate input and gate tensors.

Usage

```
ggml_geglu_split(ctx, a, b)
```

Arguments

ctx	GGML context
a	Input tensor (the values to be gated)
b	Gate tensor (same shape as a)

Details

Formula: output = GELU(a) * b

Value

Tensor with same shape as input tensors

ggml_gelu *GELU Activation (Graph)*

Description

Creates a graph node for GELU (Gaussian Error Linear Unit) activation. CRITICAL for GPT models.

Usage

```
ggml_gelu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the GELU operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
result <- ggml_gelu(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

---

ggml_gelu_back *GELU Backward (Graph)*

Description

Gradient of the GELU activation: `dx = dy * dgelu(x)`. Wired into the autodiff engine, so `ggml_compile(activation = "gelu")` trains – this is a ggmlR extension, upstream ggml has no GELU backward and aborts instead.

Usage

```
ggml_gelu_back(ctx, a, b)
```

Arguments

`ctx`	GGML context
`a`	Gradient tensor from upstream (`dy`)
`b`	Forward input tensor (`x`)

Details

The derivative is that of the tanh approximation `ggml_gelu` computes, `g(x) = 0.5x(1 + tanh(u))` with `u = c*x*(1 + a*x^2)`. The quick and erf variants (`ggml_gelu_quick`, `ggml_gelu_erf`) are different functions with different derivatives and remain without a backward rule.

Value

Gradient tensor for the input

See Also

`ggml_gelu`, `ggml_silu_back`

ggml_gelu_erf *Exact GELU Activation (Graph)*

Description

Creates a graph node for exact GELU using the error function (erf). GELU(x) = x * 0.5 * (1 + erf(x / sqrt(2))). More accurate than approximate GELU but potentially slower on some backends.

Usage

```
ggml_gelu_erf(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the exact GELU operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
r <- ggml_gelu_erf(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)
ggml_free(ctx)
```

ggml_gelu_inplace *GELU Activation In-place (Graph)*

Description

Creates a graph node for in-place GELU (Gaussian Error Linear Unit) activation. CRITICAL for GPT models with memory efficiency.

Usage

```
ggml_gelu_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with GELU applied

---

ggml_gelu_quick *GELU Quick Activation (Graph)*

Description

Creates a graph node for fast approximation of GELU. Faster than standard GELU with minimal accuracy loss.

Usage

```
ggml_gelu_quick(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the GELU quick operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
result <- ggml_gelu_quick(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)
ggml_free(ctx)
```

ggml_get_f32 *Get F32 data*

Description

Get F32 data

Get F32 Data

Usage

```
ggml_get_f32(tensor)

ggml_get_f32(tensor)
```

Arguments

tensor	Tensor

Value

Numeric vector with tensor values

Numeric vector

Examples

```
ctx <- ggml_init(1024 * 1024)
tensor <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(tensor, c(1, 2, 3, 4, 5))
ggml_get_f32(tensor)
ggml_free(ctx)


ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(t, c(1, 2, 3, 4, 5))
ggml_get_f32(t)
ggml_free(ctx)
```

ggml_get_f32_nd *Get Single Float Value by N-D Index*

Description

Gets a single f32 value from the tensor at position [i0, i1, i2, i3]. Works with any tensor type (auto-converts to float).

Usage

```
ggml_get_f32_nd(tensor, i0, i1 = 0, i2 = 0, i3 = 0)
```

Arguments

tensor	Tensor pointer
i0, i1, i2, i3	Indices (0-based)

Value

Float value

ggml_get_first_tensor *Get First Tensor from Context*

Description

Get First Tensor from Context

Usage

```
ggml_get_first_tensor(ctx)
```

Arguments

ctx	GGML context

Value

Tensor pointer or NULL

ggml_get_i32 *Get I32 Data*

Description

Gets integer data from an I32 tensor (e.g., from ggml_argmax)

Usage

```
ggml_get_i32(tensor)
```

Arguments

tensor	Tensor of type GGML_TYPE_I32

Value

Integer vector

Examples

```
ctx <- ggml_init(1024 * 1024)
pos <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 10)
ggml_set_i32(pos, 0:9)
ggml_get_i32(pos)
ggml_free(ctx)
```

ggml_get_i32_nd *Get Single Int32 Value by N-D Index*

Description

Gets a single i32 value from the tensor at position [i0, i1, i2, i3].

Usage

```
ggml_get_i32_nd(tensor, i0, i1 = 0, i2 = 0, i3 = 0)
```

Arguments

tensor	Tensor pointer
i0, i1, i2, i3	Indices (0-based)

Value

Integer value

ggml_get_layer *Get a Layer from a Sequential Model*

Description

Retrieves a layer by name or by integer index (1-based).

Usage

```
ggml_get_layer(model, index = NULL, name = NULL)
```

Arguments

model	A ggml_sequential_model object
index	Integer index of the layer (1-based), or NULL
name	Character name of the layer, or NULL

Value

The layer list object

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(64, activation = "relu", name = "hidden") |>
  ggml_layer_dense(10, activation = "softmax", name = "output")

ggml_get_layer(model, index = 1)
ggml_get_layer(model, name = "output")
```

ggml_get_max_tensor_size *Get Maximum Tensor Size*

Description

Returns the maximum tensor size that can be allocated in the context

Usage

```
ggml_get_max_tensor_size(ctx)
```

Arguments

ctx	GGML context

Value

Maximum tensor size in bytes

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_get_max_tensor_size(ctx)
ggml_free(ctx)
```

---

ggml_get_mem_size *Get Context Memory Size*

Description

Returns the total memory pool size of the context

Usage

```
ggml_get_mem_size(ctx)
```

Arguments

ctx	GGML context

Value

Total memory size in bytes

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_get_mem_size(ctx)
ggml_free(ctx)
```

ggml_get_n_threads *Get Number of Threads*

Description

Get the current number of threads for GGML operations

Usage

```
ggml_get_n_threads()
```

Value

Number of threads

Examples

```
ggml_get_n_threads()
```

ggml_get_name *Get Tensor Name*

Description

Retrieves the name of a tensor.

Usage

```
ggml_get_name(tensor)
```

Arguments

tensor	Tensor pointer

Value

Character string name or NULL if not set

ggml_get_next_tensor *Get Next Tensor from Context*

Description

Get Next Tensor from Context

Usage

```
ggml_get_next_tensor(ctx, tensor)
```

Arguments

ctx	GGML context
tensor	Current tensor

Value

Next tensor pointer or NULL

ggml_get_no_alloc *Get No Allocation Mode*

Description

Check if no-allocation mode is enabled

Usage

```
ggml_get_no_alloc(ctx)
```

Arguments

ctx	GGML context

Value

Logical indicating if no_alloc is enabled

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_get_no_alloc(ctx)
ggml_free(ctx)
```

ggml_get_op_params *Get Tensor Operation Parameters*

Description

Returns the raw op_params bytes from a tensor. These parameters control operation-specific behavior (e.g., precision, mode).

Usage

```
ggml_get_op_params(tensor)
```

Arguments

tensor	External pointer to tensor

Value

Raw vector of op_params bytes

See Also

Other tensor: ggml_are_same_layout(), ggml_get_op_params_f32(), ggml_get_op_params_i32(), ggml_set_op_params(), ggml_set_op_params_f32(), ggml_set_op_params_i32()

ggml_get_op_params_f32 *Get Float Op Parameter*

Description

Gets a single float value from tensor op_params at given index.

Usage

```
ggml_get_op_params_f32(tensor, index)
```

Arguments

tensor	External pointer to tensor
index	0-based index (0-15 for 64-byte op_params)

Value

Numeric value

See Also

Other tensor: ggml_are_same_layout(), ggml_get_op_params(), ggml_get_op_params_i32(), ggml_set_op_params(), ggml_set_op_params_f32(), ggml_set_op_params_i32()

`ggml_get_op_params_i32` *Get Integer Op Parameter*

Description

Gets a single int32 value from tensor op_params at given index.

Usage

```
ggml_get_op_params_i32(tensor, index)
```

Arguments

`tensor`	External pointer to tensor
`index`	0-based index (0-15 for 64-byte op_params)

Value

Integer value

See Also

Other tensor: ggml_are_same_layout(), ggml_get_op_params(), ggml_get_op_params_f32(), ggml_set_op_params(), ggml_set_op_params_f32(), ggml_set_op_params_i32()

`ggml_get_rel_pos` *Get Relative Position (Graph)*

Description

Gathers relative-position rows for relative-position attention bias.

Usage

```
ggml_get_rel_pos(ctx, a, qh, kh)
```

Arguments

ctx	GGML context
a	Input tensor
qh	Query height
kh	Key height

Value

Relative-position tensor

---

ggml_get_rows *Get Rows by Indices (Graph)*

Description

Creates a graph node that extracts rows from a tensor by index. This is commonly used for embedding lookup in LLMs.

Usage

```
ggml_get_rows(ctx, a, b)
```

Arguments

ctx	GGML context
a	Data tensor of shape [n_embd, n_rows, ...] - the embedding table
b	Index tensor (int32) of shape [n_indices] - which rows to extract

Details

This operation is fundamental for embedding lookup in transformers: given a vocabulary embedding matrix and token indices, it retrieves the corresponding embedding vectors.

Value

Tensor of shape [n_embd, n_indices, ...] containing the selected rows

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Create embedding matrix: 10 tokens, 4-dim embeddings
embeddings <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 10)
ggml_set_f32(embeddings, rnorm(40))
# Token indices to look up
indices <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 3)
ggml_set_i32(indices, c(0L, 5L, 2L))
# Get embeddings for tokens 0, 5, 2
```

```
result <- ggml_get_rows(ctx, embeddings, indices)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_get_rows_back *Get Rows Backward (Graph)*

Description

Backward pass for ggml_get_rows operation. Accumulates gradients at the original row positions.

Usage

```
ggml_get_rows_back(ctx, a, b, c)
```

Arguments

ctx	GGML context
a	Gradient of get_rows output
b	Index tensor (same as forward pass)
c	Reference tensor defining output shape

Value

Gradient tensor for the embedding matrix

---

ggml_get_unary_op *Get Unary Operation from Tensor*

Description

Returns the unary operation type for a unary operation tensor.

Usage

```
ggml_get_unary_op(tensor)
```

Arguments

tensor	Tensor pointer (must be a unary operation result)

Value

Integer unary operation type

See Also

Other op_info: ggml_op_desc(), ggml_op_name(), ggml_op_symbol(), ggml_unary_op_name()

---

ggml_glu *Generic GLU (Gated Linear Unit) (Graph)*

Description

Creates a graph node for GLU operation with specified gating type. GLU splits the input tensor in half along the first dimension, applies an activation to the first half (x), and multiplies it with the second half (gate).

Usage

```
ggml_glu(ctx, a, op, swapped = FALSE)
```

Arguments

ctx	GGML context
a	Input tensor (first dimension must be even)
op	GLU operation type (GGML_GLU_OP_REGLU, GGML_GLU_OP_GEGLU, etc.)
swapped	If TRUE, swap x and gate halves (default FALSE)

Details

Formula: output = activation(x) * gate where x and gate are the two halves of the input tensor.

Value

Tensor with shape [n/2, ...] where n is the first dimension of input

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Create tensor with 10 columns (will be split into 5 + 5)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 4)
ggml_set_f32(a, rnorm(40))
# Apply SwiGLU
r <- ggml_glu(ctx, a, GGML_GLU_OP_SWIGLU, FALSE)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # Shape: 5x4
ggml_free(ctx)
```

GGML_GLU_OP_REGLU *GLU Operation Types*

Description

Constants for GLU (Gated Linear Unit) operation types. Used with ggml_glu() and ggml_glu_split().

Usage

```
GGML_GLU_OP_REGLU

GGML_GLU_OP_GEGLU

GGML_GLU_OP_SWIGLU

GGML_GLU_OP_SWIGLU_OAI

GGML_GLU_OP_GEGLU_ERF

GGML_GLU_OP_GEGLU_QUICK
```

Format

Integer constants

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

Details

- `GGML_GLU_OP_REGLU` (0): ReGLU - ReLU gating
- `GGML_GLU_OP_GEGLU` (1): GeGLU - GELU gating (used in GPT-NeoX, Falcon)
- `GGML_GLU_OP_SWIGLU` (2): SwiGLU - SiLU/Swish gating (used in LLaMA, Mistral)
- `GGML_GLU_OP_SWIGLU_OAI` (3): SwiGLU OpenAI variant
- `GGML_GLU_OP_GEGLU_ERF` (4): GeGLU with exact erf implementation
- `GGML_GLU_OP_GEGLU_QUICK` (5): GeGLU with fast approximation

Value

An integer constant representing a GLU operation type

Examples

```
GGML_GLU_OP_REGLU       # 0 - ReLU gating
GGML_GLU_OP_GEGLU       # 1 - GELU gating
GGML_GLU_OP_SWIGLU      # 2 - SiLU/Swish gating
GGML_GLU_OP_SWIGLU_OAI  # 3 - SwiGLU OpenAI
GGML_GLU_OP_GEGLU_ERF   # 4 - GELU with erf
GGML_GLU_OP_GEGLU_QUICK # 5 - Fast GELU
```

---

ggml_glu_split *Generic GLU Split (Graph)*

Description

Creates a graph node for GLU with separate input and gate tensors. Unlike standard GLU which splits a single tensor, this takes two separate tensors.

Usage

```
ggml_glu_split(ctx, a, b, op)
```

Arguments

ctx	GGML context
a	Input tensor (the values to be gated)
b	Gate tensor (same shape as a)
op	GLU operation type (GGML_GLU_OP_REGLU, GGML_GLU_OP_GEGLU, etc.)

Value

Tensor with same shape as input tensors

---

ggml_graph_add_node *Append a Node to a Computation Graph*

Description

Adds `tensor` to `graph` as a node, without expanding its dependencies. For the usual case – adding a tensor together with everything it depends on – use `ggml_graph_expand`.

Usage

```
ggml_graph_add_node(graph, tensor)
```

Arguments

graph	Graph to add to
tensor	Tensor to append

Value

NULL, invisibly; graph is modified in place

See Also

ggml_graph_expand

---

ggml_graph_clear *Clear a Computation Graph*

Description

Removes every node from graph, keeping its allocated capacity so it can be rebuilt without a fresh allocation.

Usage

```
ggml_graph_clear(graph)
```

Arguments

graph	Graph to clear

Value

NULL, invisibly; graph is modified in place

---

ggml_graph_compute *Compute graph*

Description

Executes all operations in the computation graph.

Executes the computation graph using CPU backend

Usage

```
ggml_graph_compute(ctx, graph)

ggml_graph_compute(ctx, graph)
```

Arguments

ctx	GGML context
graph	Graph object created by ggml_build_forward_expand

Value

NULL (invisible)

No return value, called for side effects

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
result <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)


ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_set_f32(a, 1:10)
ggml_set_f32(b, 11:20)
c <- ggml_add(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, c)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(c)
ggml_free(ctx)
```

---

ggml_graph_compute_with_ctx *Compute Graph with Context (Alternative Method)*

Description

Computes the computation graph using the context-based method. This is an alternative to ggml_graph_compute() that uses ggml_graph_plan() and ggml_graph_compute() internally.

Usage

```
ggml_graph_compute_with_ctx(ctx, graph, n_threads = 0L)
```

Arguments

ctx	GGML context
graph	Graph object created by ggml_build_forward_expand
n_threads	Number of threads to use (0 for auto-detect, default: 0)

Value

No return value, called for side effects

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_set_f32(a, 1:10)
c <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, c)
ggml_graph_compute_with_ctx(ctx, graph)
result <- ggml_get_f32(c)
ggml_free(ctx)
```

---

ggml_graph_cpy *Copy One Graph's Nodes Into Another*

Description

Copies the contents of src into dst, which must already have room for them. Unlike ggml_graph_dup this allocates nothing.

Usage

```
ggml_graph_cpy(src, dst)
```

Arguments

src	Source graph
dst	Destination graph

Value

NULL, invisibly; dst is modified in place

ggml_graph_dump_dot *Export Graph to DOT Format*

Description

Exports the computation graph to a DOT file for visualization. The DOT file can be converted to an image using Graphviz tools.

Usage

```
ggml_graph_dump_dot(graph, leafs = NULL, filename)
```

Arguments

graph	Graph object
leafs	Optional graph with leaf tensors (NULL for none)
filename	Output filename (should end with .dot)

Value

No return value, called for side effects

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, b)
ggml_graph_dump_dot(graph, NULL, tempfile(fileext = ".dot"))
ggml_free(ctx)
```

ggml_graph_dup *Duplicate a Computation Graph*

Description

Copies graph into ctx. With force_grads = TRUE the copy carries gradient storage even when the original does not, which is one way to turn a plain forward graph into one that ggml_build_backward_expand accepts.

Usage

```
ggml_graph_dup(ctx, graph, force_grads = FALSE)
```

Arguments

ctx	GGML context to allocate the copy in
graph	Graph to duplicate
force_grads	Logical; give the copy gradient storage (default FALSE)

Value

A new graph object (external pointer)

See Also

ggml_build_forward_expand_grads

---

ggml_graph_expand *Add Another Root to an Existing Computation Graph*

Description

ggml_build_forward_expand() always creates a fresh graph, so it can only express a single root. A model with several *independent* output branches needs every output expanded into the same graph: an output that is unreachable from the first root never enters the graph, the scheduler never assigns it a buffer, and reading it back fails. This appends tensor and its ancestors to a graph that already exists.

Usage

```
ggml_graph_expand(graph, tensor)
```

Arguments

graph	Graph object returned by ggml_build_forward_expand().
tensor	Additional output tensor to expand into graph.

Value

NULL, invisibly. graph is modified in place.

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, 1:4)
ggml_set_f32(b, 5:8)
# Two independent outputs -- neither reachable from the other.
o1 <- ggml_add(ctx, a, a)
o2 <- ggml_add(ctx, b, b)
```

```
graph <- ggml_build_forward_expand(ctx, o1)
ggml_graph_expand(graph, o2)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_graph_get_grad *Get the Gradient Tensor of a Node*

Description

Returns the tensor holding d(output)/d(node) after ggml_build_backward_expand has added the backward pass and the graph has been computed.

Usage

```
ggml_graph_get_grad(graph, node)

ggml_graph_get_grad_acc(graph, node)
```

Arguments

graph	Graph carrying a backward pass
node	Tensor whose gradient is wanted

Details

Returns NULL when the node has no gradient in this graph – because it was never marked with ggml_set_param, or because nothing differentiable connects it to the output.

Value

The gradient tensor, or NULL

For ggml_graph_get_grad_acc(), the gradient accumulator tensor, or NULL

See Also

ggml_build_backward_expand

ggml_graph_get_tensor *Get Tensor from Graph by Name*

Description

Finds a tensor in the computation graph by its name

Usage

```
ggml_graph_get_tensor(graph, name)
```

Arguments

graph	Graph object
name	Character string with tensor name

Value

Tensor pointer or NULL if not found

ggml_graph_n_nodes *Get Number of Nodes in Graph*

Description

Returns the number of computation nodes in the graph

Usage

```
ggml_graph_n_nodes(graph)
```

Arguments

graph	Graph object

Value

Integer number of nodes

ggml_graph_node *Get Graph Node*

Description

Gets a specific node (tensor) from the computation graph by index

Usage

```
ggml_graph_node(graph, i)
```

Arguments

graph	Graph object
i	Node index (0-based, negative indices count from end)

Value

Tensor pointer

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_add(ctx, a, a)
graph <- ggml_build_forward_expand(ctx, b)
# Get the last node (output)
output <- ggml_graph_node(graph, -1)
ggml_free(ctx)
```

ggml_graph_overhead *Get Graph Overhead*

Description

Returns the memory overhead required for a computation graph

Usage

```
ggml_graph_overhead()
```

Value

Size in bytes

ggml_graph_print *Print Graph Information*

Description

Prints debug information about the computation graph

Usage

```
ggml_graph_print(graph)
```

Arguments

graph	Graph object

Value

No return value, called for side effects

ggml_graph_reset *Reset Graph (for backpropagation)*

Description

Zeroes every gradient in the graph and seeds the loss node's own gradient with 1 – the value the chain rule starts from. **Call this after** ggml_build_backward_expand **and before** ggml_graph_compute: without it every gradient computes as zero, since the backward pass multiplies through a seed that was never set.

Usage

```
ggml_graph_reset(graph)
```

Arguments

graph	Graph object with gradients allocated

Details

Requires a graph built by ggml_build_forward_expand_grads; an inference-only graph has no gradients to reset and is rejected.

Value

No return value, called for side effects

See Also

ggml_build_backward_expand, ggml_build_forward_expand_grads

---

ggml_graph_view *Create a View of a Subgraph*

Description

Creates a view of a portion of a computation graph, containing nodes from index i0 to i1 (exclusive). The view shares the underlying nodes but does not include leaf tensors or gradients.

Usage

```
ggml_graph_view(graph, i0, i1)
```

Arguments

graph	External pointer to computation graph
i0	Start index (0-based, inclusive)
i1	End index (exclusive)

Value

External pointer to graph view

See Also

Other graph: ggml_op_can_inplace()

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
b <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, b)
n_nodes <- ggml_graph_n_nodes(graph)
view <- ggml_graph_view(graph, 0, n_nodes)
ggml_free(ctx)
```

ggml_group_norm *Group Normalization (Graph)*

Description

Creates a graph node for group normalization. Normalizes along ne0*ne1*n_groups dimensions. Used in Stable Diffusion and other image generation models.

Usage

```
ggml_group_norm(ctx, a, n_groups, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Input tensor
n_groups	Number of groups to divide channels into
eps	Epsilon for numerical stability (default 1e-5)

Value

Tensor representing the group norm operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# 4 channels, 2 groups (2 channels per group)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 8)
ggml_set_f32(a, rnorm(32))
result <- ggml_group_norm(ctx, a, n_groups = 2)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_group_norm_inplace *Group Normalization In-place (Graph)*

Description

Creates a graph node for in-place group normalization.

Usage

```
ggml_group_norm_inplace(ctx, a, n_groups, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)
n_groups	Number of groups
eps	Epsilon for numerical stability (default 1e-5)

Value

View of input tensor with group norm applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 8)
ggml_set_f32(a, rnorm(32))
result <- ggml_group_norm_inplace(ctx, a, n_groups = 2)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_gru *Create a GRU Layer Object*

Description

Create a GRU Layer Object

Usage

```
ggml_gru(
  units,
  return_sequences = FALSE,
  activation = "tanh",
  recurrent_activation = "sigmoid",
  name = NULL,
  trainable = TRUE
)
```

Arguments

units	Integer, number of hidden units.
return_sequences	Logical.
activation	Candidate activation (default "tanh").

recurrent_activation
: Gate activation (default "sigmoid").

name
: Optional character name.

trainable
: Logical.

Value

A ggml_layer object.

---

ggml_hardsigmoid *Hard Sigmoid Activation (Graph)*

Description

Creates a graph node for Hard Sigmoid activation. HardSigmoid(x) = ReLU6(x + 3) / 6 = min(max(0, x + 3), 6) / 6. A computationally efficient approximation of the sigmoid function.

Usage

```
ggml_hardsigmoid(ctx, a)
```

Arguments

ctx
: GGML context

a
: Input tensor

Value

Tensor representing the Hard Sigmoid operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-4, -1, 0, 1, 4))
r <- ggml_hardsigmoid(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # [0, 0.333, 0.5, 0.667, 1]
ggml_free(ctx)
```

ggml_hardswish *Hard Swish Activation (Graph)*

Description

Creates a graph node for Hard Swish activation. HardSwish(x) = x * ReLU6(x + 3) / 6 = x * min(max(0, x + 3), 6) / 6. Used in MobileNetV3 and other efficient architectures.

Usage

```
ggml_hardswish(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the Hard Swish operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-4, -1, 0, 1, 4))
r <- ggml_hardswish(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)
ggml_free(ctx)
```

ggml_im2col *Image to Column (Graph)*

Description

Transforms image data into column format for efficient convolution. This is a low-level operation used internally by convolution implementations.

Usage

```
ggml_im2col(
  ctx,
  a,
  b,
  s0,
  s1,
  p0,
  p1,
  d0,
  d1,
  is_2D = TRUE,
  dst_type = GGML_TYPE_F16
)
```

Arguments

ctx	GGML context
a	Convolution kernel tensor
b	Input data tensor
s0	Stride dimension 0
s1	Stride dimension 1
p0	Padding dimension 0
p1	Padding dimension 1
d0	Dilation dimension 0
d1	Dilation dimension 1
is_2D	Whether this is a 2D operation (default TRUE)
dst_type	Output type (default GGML_TYPE_F16)

Value

Transformed tensor in column format

---

ggml_init *Initialize GGML context*

Description

Initialize GGML context

Usage

```
ggml_init(mem_size = 16 * 1024 * 1024, no_alloc = FALSE)
```

Arguments

mem_size	Memory size in bytes
no_alloc	If TRUE, don't allocate memory for tensors (default: FALSE)

Value

GGML context pointer

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_free(ctx)
```

---

ggml_init_auto *Create Context with Auto-sizing*

Description

Creates a context with automatically calculated size based on planned tensors.

Usage

```
ggml_init_auto(..., extra_mb = 10, type = GGML_TYPE_F32, no_alloc = FALSE)
```

Arguments

...	Named arguments with tensor dimensions (integer vectors)
extra_mb	Extra megabytes to add (default: 10)
type	Tensor type (default: GGML_TYPE_F32)
no_alloc	If TRUE, don't allocate memory for tensors (default: FALSE)

Value

GGML context

Examples

```
ctx <- ggml_init_auto(mat1 = c(1000L, 1000L), mat2 = c(1000L, 1000L))
ggml_free(ctx)
```

ggml_inject *Inject a single-cell result back into its container*

Description

Writes a ggml_result back into the appropriate slot of a Seurat or SingleCellExperiment object, returning the updated object. The destination depends on result$metadata$kind: a dimensionality reduction (the default) goes into a DimReduc; a "transform" (normalize / scale) overwrites an assay layer; a "graph" (neighbors) writes <assay>_nn and <assay>_snn Graph objects into @graphs, exactly where Seurat's FindNeighbors puts them so FindClusters can consume them. Component standard deviations and the backend used are recorded alongside so downstream tools and the user can see how the result was produced.

Usage

```
ggml_inject(
  x,
  result,
  reduction_name = "ggml",
  key = "GGML_",
  assay = NULL,
  ...
)

## S3 method for class 'Seurat'
ggml_inject(
  x,
  result,
  reduction_name = "ggml",
  key = "GGML_",
  assay = NULL,
  ...
)

## S3 method for class 'SingleCellExperiment'
ggml_inject(
  x,
  result,
  reduction_name = "ggml",
  key = "GGML_",
  assay = NULL,
  ...
)
```

Arguments

x	A Seurat object (the one the data was extracted from).

result	A ggml_result.
reduction_name	Name of the reduction slot to create, e.g. "ggml" (default). For Seurat this becomes x[["ggml"]].
key	Column-name prefix for the embedding, e.g. "GGML_".
assay	Assay to associate the reduction with (Seurat). Defaults to the object's default assay.
...	Passed to methods.

Value

The updated container.

---

ggml_input *Declare a Functional API Input Tensor*

Description

Creates a symbolic input node for the Functional API. The node records only the *shape* of one sample (without batch dimension); actual memory is allocated when ggml_compile() is called.

Usage

```
ggml_input(shape, name = NULL, dtype = "float32")
```

Arguments

shape	Integer vector describing the shape of a single sample. For flat feature vectors use a scalar, e.g. shape = 64L. For 2-D inputs (sequences) use c(length, channels). For 3-D inputs (images) use c(H, W, C).
name	Optional character name for the input tensor.
dtype	Data type of the input: "float32" (default) or "int32" (for embedding/token-index inputs).

Value

A ggml_tensor_node object.

Examples

```
x <- ggml_input(shape = 64L)
x <- ggml_input(shape = c(28L, 28L, 1L), name = "image")
x <- ggml_input(shape = 10L, dtype = "int32")  # token indices
```

ggml_is_available *Check if GGML is available*

Description

Check if GGML is available

Usage

```
ggml_is_available()
```

Value

TRUE if GGML library is loaded

Examples

```
ggml_is_available()
```

ggml_is_contiguous *Check if Tensor is Contiguous*

Description

Returns TRUE if tensor data is stored contiguously in memory

Usage

```
ggml_is_contiguous(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_is_contiguous(t)
ggml_free(ctx)
```

ggml_is_contiguous_0 *Check Tensor Contiguity (Dimension 0)*

Description

Check if tensor is contiguous. Same as `ggml_is_contiguous`.

Usage

```
ggml_is_contiguous_0(tensor)
```

Arguments

`tensor`	Tensor pointer

Value

Logical indicating contiguity

See Also

Other tensor_layout: `ggml_are_same_stride()`, `ggml_can_repeat()`, `ggml_count_equal()`, `ggml_is_contiguous_1()`, `ggml_is_contiguous_2()`, `ggml_is_contiguous_channels()`, `ggml_is_contiguous_rows()`, `ggml_is_contiguously_allocated()`

ggml_is_contiguous_1 *Check Tensor Contiguity (Dimensions >= 1)*

Description

Check if tensor is contiguous for dimensions >= 1. Allows non-contiguous first dimension.

Usage

```
ggml_is_contiguous_1(tensor)
```

Arguments

`tensor`	Tensor pointer

Value

Logical indicating contiguity for dims >= 1

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_can_repeat(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_2(), ggml_is_contiguous_channels(), ggml_is_contiguous_rows(), ggml_is_contiguously_allocated()

---

ggml_is_contiguous_2 *Check Tensor Contiguity (Dimensions >= 2)*

Description

Check if tensor is contiguous for dimensions >= 2. Allows non-contiguous first two dimensions.

Usage

```
ggml_is_contiguous_2(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical indicating contiguity for dims >= 2

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_can_repeat(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_channels(), ggml_is_contiguous_rows(), ggml_is_contiguously_allocated()

---

ggml_is_contiguous_channels *Check Channel-wise Contiguity*

Description

Check if tensor has contiguous channels (important for CNN operations). Data for each channel should be stored contiguously.

Usage

```
ggml_is_contiguous_channels(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical indicating channel-wise contiguity

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_can_repeat(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_2(), ggml_is_contiguous_rows(), ggml_is_contiguously_alloca

---

ggml_is_contiguous_rows

Check Row-wise Contiguity

Description

Check if tensor has contiguous rows (important for matrix operations). Each row should be stored contiguously in memory.

Usage

```
ggml_is_contiguous_rows(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical indicating row-wise contiguity

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_can_repeat(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_2(), ggml_is_contiguous_channels(), ggml_is_contiguously_al

---

ggml_is_contiguously_allocated

Check If Tensor is Contiguously Allocated

Description

Check if tensor data is contiguously allocated in memory. Different from contiguous layout - this checks the actual allocation.

Usage

```
ggml_is_contiguously_allocated(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical indicating if data is contiguously allocated

See Also

Other tensor_layout: ggml_are_same_stride(), ggml_can_repeat(), ggml_count_equal(), ggml_is_contiguous_0(), ggml_is_contiguous_1(), ggml_is_contiguous_2(), ggml_is_contiguous_channels(), ggml_is_contiguous_rows(

---

ggml_is_permuted *Check if Tensor is Permuted*

Description

Returns TRUE if tensor dimensions have been permuted

Usage

```
ggml_is_permuted(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_is_permuted(t)
ggml_free(ctx)
```

ggml_is_quantized *Check If Type is Quantized*

Description

Returns TRUE if the GGML type is a quantized format.

Usage

```
ggml_is_quantized(type)
```

Arguments

type	GGML type constant

Value

Logical indicating if type is quantized

See Also

Other type_system: ggml_blck_size(), ggml_ftype_to_ggml_type(), ggml_type_name(), ggml_type_sizef()

Examples

```
ggml_is_quantized(GGML_TYPE_F32)  # FALSE
ggml_is_quantized(GGML_TYPE_Q4_0) # TRUE
```

ggml_is_transposed *Check if Tensor is Transposed*

Description

Returns TRUE if tensor has been transposed

Usage

```
ggml_is_transposed(tensor)
```

Arguments

tensor	Tensor pointer

Value

Logical

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_is_transposed(t)
ggml_free(ctx)
```

`ggml_l2_norm` *L2 Normalization (Graph)*

Description

Creates a graph node for L2 normalization (unit norm). Normalizes vectors to unit length: x / ||x||_2. Used in RWKV v7 and embedding normalization.

Usage

```
ggml_l2_norm(ctx, a, eps = 1e-05)
```

Arguments

`ctx`	GGML context
`a`	Input tensor
`eps`	Epsilon for numerical stability (default 1e-5)

Value

Tensor representing the L2 norm operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(3, 0, 0, 4))  # Length = 5
result <- ggml_l2_norm(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [0.6, 0, 0, 0.8] unit vector
ggml_free(ctx)
```

ggml_l2_norm_inplace *L2 Normalization In-place (Graph)*

Description

Creates a graph node for in-place L2 normalization.

Usage

```
ggml_l2_norm_inplace(ctx, a, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)
eps	Epsilon for numerical stability (default 1e-5)

Value

View of input tensor with L2 norm applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(3, 0, 0, 4))
result <- ggml_l2_norm_inplace(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_layer_add *Element-wise Addition of Two Tensor Nodes*

Description

Adds two (or more) tensor nodes element-wise. All tensors must have the same shape. This is the functional equivalent of a residual / skip connection.

Usage

```
ggml_layer_add(tensors, name = NULL)
```

Arguments

tensors	A list of ggml_tensor_node objects (length >= 2).
name	Optional character name for the layer.

Value

A new ggml_tensor_node representing the sum.

Examples

```
x    <- ggml_input(shape = 64L)
a    <- x |> ggml_layer_dense(64, activation = "relu")
b    <- x |> ggml_layer_dense(64)
out  <- ggml_layer_add(list(a, b))
```

---

ggml_layer_attention *Add a Multi-Head Attention Layer*

Description

Scaled dot-product attention with n_heads heads, in the pipe-style form that appends a layer to a functional graph. For the reusable *layer object* that shares one set of weights between several applications – how an encoder block is reused across a stack – see ggml_attention.

Usage

```
ggml_layer_attention(
  x,
  d_model,
  n_heads = 1L,
  causal = FALSE,
  context = NULL,
  mask = NULL,
  rope = FALSE,
  rope_base = 10000,
  dropout = 0,
  bias = TRUE,
  name = NULL,
  trainable = TRUE
)
```

Arguments

x	A ggml_tensor_node of shape c(seq_len, d_model), or a list of two such nodes list(query, context) for cross-attention.
d_model	Integer, model width. Must be divisible by n_heads and match the parent's feature dimension.
n_heads	Integer, number of attention heads (default 1).
causal	Logical: mask out positions after the current one, making the layer autoregressive (default FALSE). Self-attention only.
context	Optional ggml_tensor_node to take keys and values from, making the layer cross-attention: x supplies the queries and decides the output length, context the keys and values. The two may differ in sequence length but must share d_model. Equivalent to passing list(query, context) as x.
mask	Optional ggml_tensor_node of shape c(seq_q, seq_kv) – one row per query, holding that query's entry for every key – added to the scores before the softmax: use 0 where a key may be attended to and a large negative value (-1e9) where it may not. This is how padding is excluded when a batch holds sequences of different lengths. Combines with causal.
rope	Logical: apply rotary position embedding to the queries and keys (default FALSE). Unlike ggml_layer_positional_embedding this has no weights and encodes RELATIVE position – the score between two tokens depends on the distance between them – so it carries over to sequence lengths the model was not trained on. Needs an even d_model / n_heads.
rope_base	Rotation base for rope (default 10000, the value
dropout	Dropout rate applied to the attention weights during training (default 0, no dropout). used by LLaMA-family models). Larger values stretch the rotation over longer contexts.
bias	Logical: add a bias to the output projection (default TRUE).
name	Optional layer name.
trainable	Logical.

Value

A new ggml_tensor_node of shape c(seq_len, d_model).

Shapes

x must be a sequence node of shape c(seq_len, d_model), the same layout ggml_layer_lstm takes. The output has that same shape, so attention blocks stack directly and a residual connection (ggml_layer_add(list(x, attn_out))) needs no reshaping.

How it is computed

All heads are computed in one batched pass rather than in a loop: the projections are reshaped to [d_head, n_heads, seq, batch] and permuted so that ggml_mul_mat() batches over the head and batch axes. The node count is therefore independent of n_heads.

Cross-attention

Pass a list of two nodes, `ggml_layer_attention(list(query, context), ...)`, to take queries from the first and keys/values from the second. The two may differ in sequence length but must share `d_model`. A single node is self-attention.

See Also

`ggml_attention` for the shared-weights layer object.

Examples

```
# A transformer encoder block: attention + residual, then a feed-forward
# sublayer + residual.
x    <- ggml_input(shape = c(10L, 32L))
attn <- x |> ggml_layer_attention(d_model = 32L, n_heads = 4L)
h    <- ggml_layer_add(list(x, attn))
ff   <- h |> ggml_layer_dense(32L, activation = "relu")
out  <- ggml_layer_add(list(h, ff))

# GPT-style causal self-attention.
dec <- x |> ggml_layer_attention(32L, n_heads = 4L, causal = TRUE)

# Padding mask, fed as a second input alongside the sequence.
pad <- ggml_input(shape = c(10L, 10L), name = "mask")
enc <- x |> ggml_layer_attention(32L, n_heads = 4L, mask = pad)
```

`ggml_layer_batch_norm` *Add Batch Normalization Layer*

Description

Applies normalization: RMS-normalizes the input, then scales by gamma and shifts by beta (both learnable). Uses `ggml_rms_norm` which supports backward pass for training.

Usage

```
ggml_layer_batch_norm(model, eps = 1e-05, name = NULL, trainable = TRUE)
```

Arguments

`model`	A ggml_sequential_model object
`eps`	Small constant for numerical stability (default 1e-5)
`name`	Optional character name for the layer.
`trainable`	Logical; whether the layer weights are updated during training.

Value

The model object with the batch_norm layer appended (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(128, input_shape = 784) |>
  ggml_layer_batch_norm() |>
  ggml_layer_dense(10, activation = "softmax")
```

---

ggml_layer_concatenate

Concatenate Tensor Nodes Along an Axis

Description

Concatenates two or more tensor nodes along the specified axis.

Usage

```
ggml_layer_concatenate(tensors, axis = 0L, name = NULL)
```

Arguments

tensors	A list of ggml_tensor_node objects (length >= 2).
axis	Integer axis along which to concatenate (0-based, ggml convention). Default 0L concatenates along the first dimension (features for flat tensors).
name	Optional character name for the layer.

Value

A new ggml_tensor_node representing the concatenated tensor.

Examples

```
x   <- ggml_input(shape = 32L)
y   <- ggml_input(shape = 32L)
out <- ggml_layer_concatenate(list(x, y), axis = 0L)
```

ggml_layer_conv_1d *Add 1D Convolution Layer*

Description

Add 1D Convolution Layer

Usage

```
ggml_layer_conv_1d(
  model,
  filters,
  kernel_size,
  activation = NULL,
  input_shape = NULL,
  strides = 1L,
  padding = "valid",
  name = NULL,
  trainable = TRUE
)
```

Arguments

model	A ggml_sequential_model object
filters	Number of output filters
kernel_size	Integer kernel size
activation	Activation function name: "relu", "sigmoid", "tanh", "softmax", or NULL
input_shape	Input shape c(L, C) - required for first layer only (length, channels)
strides	Integer stride (default 1)
padding	"valid" (no padding) or "same" (preserve length)
name	Optional character name for the layer.
trainable	Logical; whether the layer weights are updated during training.

Value

The model object with the conv_1d layer appended (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_1d(32, 3, activation = "relu",
                     input_shape = c(100, 1))
```

ggml_layer_conv_2d *Add 2D Convolution Layer*

Description

Add 2D Convolution Layer

Usage

```
ggml_layer_conv_2d(
  model,
  filters,
  kernel_size,
  activation = NULL,
  input_shape = NULL,
  strides = c(1L, 1L),
  padding = "valid",
  name = NULL,
  trainable = TRUE
)
```

Arguments

model	A ggml_sequential_model object
filters	Number of output filters
kernel_size	Integer or vector of 2 integers for kernel height and width
activation	Activation function name: "relu", "sigmoid", "tanh", "softmax", or NULL
input_shape	Input shape c(H, W, C) - required for first layer only
strides	Integer or vector of 2 integers for stride
padding	"valid" (no padding) or "same" (preserve spatial dims)
name	Optional character name for the layer.
trainable	Logical; whether the layer weights are updated during training.

Value

The model object with the conv_2d layer appended (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1))
```

ggml_layer_custom *Define a Custom Layer From an R Function*

Description

Turns an arbitrary composition of ggml operations into a reusable functional layer. `ggml_layer_custom()` does not build anything itself — it returns a *layer function* that behaves like any other `ggml_layer_*()` and can be piped into a model.

Usage

```
ggml_layer_custom(forward, name = NULL, output_shape = NULL)
```

Arguments

`forward`	A function of `(ctx, x)` returning a ggml tensor, where `ctx` is the compute context and `x` the parent tensor. All ggml operations take `ctx` as their first argument.
`name`	Optional character name for the layer. Used as the base for auto-generated node names.
`output_shape`	Output shape excluding the batch dimension. Defaults to `NULL`, meaning the layer preserves its input shape (correct for element-wise layers such as custom activations). Supply an integer vector when `forward` changes the shape, or a function of the input shape.

Details

The `forward` function is called once while the compute graph is being built, with the active `ggml_context` and the parent tensor. Because the returned tensor becomes an ordinary graph node, gradients flow through a custom layer automatically — as long as every operation used inside `forward` implements a backward pass.

Value

A layer function of `(x, name = NULL)` that appends a custom node to the graph and returns the new `ggml_tensor_node`.

Weights

Custom layers are stateless: `forward` maps activations to activations and owns no trainable parameters. Put learnable weights in a neighbouring `ggml_layer_dense()`, or use the autograd API (`ag_*`) when a layer needs its own parameters.

Examples

```
# Mish activation: x * tanh(softplus(x))
layer_mish <- ggml_layer_custom(
  name    = "mish",
  forward = function(ctx, x) ggml_mul(ctx, x, ggml_tanh(ctx, ggml_softplus(ctx, x)))
)

inp <- ggml_input(shape = 8L)
out <- inp |>
  ggml_layer_dense(16L) |>
  layer_mish() |>
  ggml_layer_dense(2L, activation = "softmax")
model <- ggml_model(inputs = inp, outputs = out)
```

---

ggml_layer_dense *Add Dense (Fully Connected) Layer*

Description

Add Dense (Fully Connected) Layer

Usage

```
ggml_layer_dense(
  model,
  units,
  activation = NULL,
  input_shape = NULL,
  name = NULL,
  trainable = TRUE,
  time_distributed = FALSE
)
```

Arguments

model	A ggml_sequential_model object
units	Number of output units
activation	Activation function name: "relu", "sigmoid", "tanh", "softmax", or NULL
input_shape	Integer or integer vector specifying the input shape (only needed for the first layer)
name	Optional character name for the layer.
trainable	Logical; whether the layer weights are updated during training.
time_distributed	Logical; apply the kernel per position of a sequence input rather than flattening it (default FALSE). See *Time-distributed application.*

Value

The model object with the dense layer appended (invisibly).

Time-distributed application

By default a dense layer flattens its input, so a sequence node of shape `c(seq_len, features)` becomes a single vector of `units` and the sequence axis is gone. With `time_distributed = TRUE` the same kernel is applied independently at every position instead, sharing one set of weights across the sequence, and the output keeps its length: `c(seq_len, units)`. This is the semantics of Keras' `TimeDistributed`, and it is what the position-wise feed-forward sublayer of a transformer block needs – without it, a dense layer after `ggml_layer_attention` would collapse every position into one vector.

It is computed as a single batched `ggml_mul_mat()` over the combined position/batch axes, not as a loop over positions.

Functional API only: a sequential model carries a flat running shape, so there is no sequence axis to distribute over.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_flatten() |>
  ggml_layer_dense(128, activation = "relu")

# Position-wise feed-forward, as in a transformer block.
x  <- ggml_input(shape = c(10L, 32L))
ff <- x |> ggml_layer_dense(64L, activation = "relu", time_distributed = TRUE)
```

---

ggml_layer_dropout *Add Dropout Layer*

Description

Applies dropout regularization. During training, multiplies all activations by `(1 - rate)` (deterministic expected-value scaling). During inference (`training = FALSE`), the layer is an identity (no change).

Usage

```
ggml_layer_dropout(
  model,
  rate,
  stochastic = FALSE,
  name = NULL,
  trainable = FALSE
)
```

Arguments

model	A ggml_sequential_model or ggml_tensor_node.
rate	Dropout rate in [0, 1). Fraction of units to "drop".
stochastic	Logical. If TRUE, use inverted dropout with a random Bernoulli mask regenerated each epoch (proper regularization). If FALSE (default), use deterministic scaling by (1 - rate) — cheaper but weaker regularization.
name	Optional layer name.
trainable	Ignored for dropout (no weights); kept for API consistency.

Value

The model with the dropout layer appended, or a new tensor node.

Difference from Keras / inverted dropout

Keras implements *inverted dropout*: during training it applies a random Bernoulli mask and scales surviving activations *up* by 1 / (1 - rate), so the expected value of each unit is preserved and no scaling is needed at inference.

This implementation uses *deterministic scaling* (multiply by (1 - rate) at training, identity at inference) — equivalent in expected value but without stochastic noise. Consequences:

- No random mask → the regularization signal is weaker (no co-adaptation breaking).
- Activations at training are scaled *down*, not up — the magnitude seen by subsequent layers differs from Keras behaviour.
- Results are fully deterministic and reproducible without setting a seed.

Note

With stochastic = TRUE the Bernoulli mask is regenerated once per epoch (not per batch), because ggml_opt_fit processes all batches inside a single C call. This is weaker than per-batch dropout but stronger than the deterministic variant.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(128, activation = "relu", input_shape = 784L) |>
  ggml_layer_dropout(0.5, stochastic = TRUE) |>
  ggml_layer_dense(10, activation = "softmax")
```

ggml_layer_embedding *Add Embedding Layer*

Description

Looks up dense vectors for integer token indices. The input must be an integer matrix of 0-based indices in `[0, vocab_size - 1]` (use `ggml_input(shape, dtype = "int32")` in Functional mode).

Usage

```
ggml_layer_embedding(
  model,
  vocab_size,
  dim,
  name = NULL,
  trainable = TRUE,
  input_shape = NULL
)
```

Arguments

`model`	A `ggml_sequential_model` or `ggml_tensor_node`.
`vocab_size`	Number of distinct tokens (vocabulary size).
`dim`	Embedding dimension (vector length per token).
`name`	Optional layer name.
`trainable`	Logical; whether embedding weights are updated during training.
`input_shape`	Sequence length, for an embedding that is the first layer of a sequential model (its input is then a matrix of token indices, one row per sample). Ignored in the functional API, where `ggml_input(dtype = "int32")` gives the shape.

Value

The model with the embedding layer appended, or a new tensor node.

Axis order

The layer's output shape is `c(seq_len, dim)` – one row per position, as in Keras – which is the same convention a declared sequence input (`ggml_input(shape = c(seq_len, features))`) uses. So the output feeds `ggml_layer_attention`, `ggml_layer_gru` and `ggml_layer_lstm` directly, with no axis juggling in between.

The underlying tensor is `[dim, seq_len, N]`: ggml is column-major, so an R shape `c(a, b)` is the tensor `[b, a, N]` throughout the package. That matters only when reading raw output tensors – `ggml_layer_flatten()` gives the same vector either way.

Index validation

Indices must be in `[0, vocab_size - 1]`. Out-of-range values cause undefined behaviour inside the ggml kernel (no bounds check is performed at the R level).

Examples

```
inp <- ggml_input(shape = 10L, dtype = "int32")
out <- inp |>
  ggml_layer_embedding(vocab_size = 1000L, dim = 32L) |>
  ggml_layer_flatten() |>
  ggml_layer_dense(10L, activation = "softmax")
model <- ggml_model(inputs = inp, outputs = out)
```

---

ggml_layer_flatten *Add Flatten Layer*

Description

Flattens the spatial dimensions into a single vector per sample.

Usage

```
ggml_layer_flatten(model, name = NULL, trainable = TRUE)
```

Arguments

`model`	A ggml_sequential_model object
`name`	Optional character name for the layer.
`trainable`	Logical; reserved for API consistency (no weights).

Value

The model object with the flatten layer appended (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_flatten()
```

ggml_layer_global_average_pooling_2d

Global Average Pooling for 2D Feature Maps

Description

Reduces a [H, W, C] feature map to [C] by averaging all spatial positions per channel. Equivalent to Keras GlobalAveragePooling2D().

Usage

```
ggml_layer_global_average_pooling_2d(model, name = NULL, trainable = TRUE)
```

Arguments

model	A ggml_sequential_model or ggml_tensor_node.
name	Optional character name for the layer.
trainable	Logical; reserved for API consistency (no weights).

Value

Updated model or a new ggml_tensor_node.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_global_average_pooling_2d() |>
  ggml_layer_dense(10, activation = "softmax")
```

ggml_layer_global_max_pooling_2d

Global Max Pooling for 2D Feature Maps

Description

Reduces a [H, W, C] feature map to [C] by taking the maximum value per channel across all spatial positions. Equivalent to Keras GlobalMaxPooling2D().

Usage

```
ggml_layer_global_max_pooling_2d(model, name = NULL, trainable = TRUE)
```

Arguments

model	A ggml_sequential_model or ggml_tensor_node.
name	Optional character name for the layer.
trainable	Logical; reserved for API consistency (no weights).

Value

Updated model or a new ggml_tensor_node.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_global_max_pooling_2d() |>
  ggml_layer_dense(10, activation = "softmax")
```

---

ggml_layer_gru *Add a GRU Layer*

Description

Gated Recurrent Unit recurrent layer. Implemented as an unrolled computation graph (BPTT).

Usage

```
ggml_layer_gru(
  model,
  units,
  return_sequences = FALSE,
  activation = "tanh",
  recurrent_activation = "sigmoid",
  input_shape = NULL,
  name = NULL,
  trainable = TRUE
)
```

Arguments

model	A ggml_sequential_model or ggml_tensor_node.
units	Integer, number of hidden units.
return_sequences	Logical; return all hidden states or only the last.
activation	Activation for the candidate hidden state ("tanh").

recurrent_activation
: Activation for z/r gates ("sigmoid").

input_shape
: Input shape c(seq_len, input_size) – required for the first layer only.

name
: Optional layer name.

trainable
: Logical.

Value

Updated model or a new ggml_tensor_node.

Weight layout

- W_zh [input_size, 2*units] — input kernel for z and r gates.
- U_zh [units, 2*units] — recurrent kernel for z and r.
- b_zh [2*units] — bias for z and r.
- W_n [input_size, units] — input kernel for candidate.
- U_n [units, units] — recurrent kernel for candidate.
- b_n [units] — bias for candidate.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_gru(64L, input_shape = c(10L, 32L)) |>
  ggml_layer_dense(10L, activation = "softmax")
```

---

ggml_layer_layer_norm *Add Layer Normalization Layer*

Description

Normalizes each sample over its own features – subtract the mean, divide by the standard deviation – then scales by gamma and shifts by beta (both learnable). Like ggml_layer_rms_norm it keeps no running statistics and does not couple the samples in a batch, but unlike it, it does centre the input. This is the normalization of the original transformer.

Usage

```
ggml_layer_layer_norm(model, eps = 1e-05, name = NULL, trainable = TRUE)
```

Arguments

model
: A ggml_sequential_model or a ggml_tensor_node (functional API).

eps
: Small constant for numerical stability (default 1e-5)

name
: Optional character name for the layer.

trainable
: Logical; whether the layer weights are updated during training.

Value

The model with the layer appended, or a new `ggml_tensor_node` in the functional API.

See Also

`ggml_layer_rms_norm`, `ggml_layer_batch_norm`

Examples

```
x <- ggml_input(shape = c(10L, 32L))
h <- x |> ggml_layer_layer_norm()
h <- h |> ggml_layer_attention(d_model = 32L, n_heads = 4L)
```

---

ggml_layer_lstm *Add an LSTM Layer*

Description

Long Short-Term Memory recurrent layer. Implemented as an unrolled computation graph (BPTT) so that ggml's automatic differentiation works without any C extensions.

Usage

```
ggml_layer_lstm(
  model,
  units,
  return_sequences = FALSE,
  activation = "tanh",
  recurrent_activation = "sigmoid",
  input_shape = NULL,
  name = NULL,
  trainable = TRUE
)
```

Arguments

`model`	A `ggml_sequential_model` or `ggml_tensor_node`.
`units`	Integer, number of hidden units.
`return_sequences`	Logical; if `TRUE` return all hidden states, otherwise return only the last hidden state.
`activation`	Activation for the cell gate (default `"tanh"`).
`recurrent_activation`	Activation for the recurrent step (default `"sigmoid"`).
`input_shape`	Input shape `c(seq_len, input_size)` – required for the first layer only.
`name`	Optional layer name.
`trainable`	Logical.

Value

Updated model or a new ggml_tensor_node.

Weight layout

- W_gates [input_size, 4*units] — input kernel for all four gates (i, f, g, o) concatenated.
- U_gates [units, 4*units] — recurrent kernel.
- b_gates [4*units] — bias.

Input / output shapes

Input: [seq_len, input_size] per sample (R row-major), or a 3-D array [N, seq_len, input_size]. In the Functional API the input node shape should be c(seq_len, input_size).

Output (Sequential): [units] per sample when return_sequences = FALSE (default), or c(seq_len, units) when return_sequences = TRUE.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_lstm(64L, input_shape = c(10L, 32L)) |>
  ggml_layer_dense(10L, activation = "softmax")
```

---

ggml_layer_max_pooling_2d

Add 2D Max Pooling Layer

Description

Add 2D Max Pooling Layer

Usage

```
ggml_layer_max_pooling_2d(
  model,
  pool_size = c(2L, 2L),
  strides = NULL,
  name = NULL,
  trainable = TRUE
)
```

Arguments

model	A ggml_sequential_model object
pool_size	Integer or vector of 2 integers for pool height and width
strides	Integer or vector of 2 integers (defaults to pool_size)
name	Optional character name for the layer.
trainable	Logical; reserved for API consistency (no weights).

Value

The model object with the max pooling layer appended (invisibly).

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_max_pooling_2d(c(2, 2))
```

---

ggml_layer_permute *Add Permute Layer*

Description

Reorder a sample's axes. The batch axis is not part of `dims` and is never moved – `dims` indexes the per-sample shape, exactly as `input_shape` does.

Usage

```
ggml_layer_permute(
  model,
  dims,
  input_shape = NULL,
  name = NULL,
  trainable = TRUE
)
```

Arguments

`model`	A ggml_sequential_model object, or a `ggml_tensor_node` for the functional API.
`dims`	Integer vector, a permutation of `seq_along(input_shape)`. Must not include the batch axis.
`input_shape`	Input shape, required for the first layer only.
`name`	Optional character name for the layer.
`trainable`	Logical; reserved for API consistency (no weights).

Value

The model object with the permute layer appended (invisibly), or a `ggml_tensor_node` in the functional API.

Argument order

`dims` follows base R's `aperm`: `dims[i]` names the *source* axis that ends up at position `i`. So `dims = c(2, 1)` means "the new first axis is the old second one".

This is deliberately the inverse of the underlying `ggml_permute`, whose arguments are *destination* positions; the layer converts between the two so that callers can think in the R idiom. Read `ggml_permute` itself as "source axis `i` goes to position `p_i`" – mixing the two conventions up silently inverts every non-trivial permutation.

Typical use

Reordering a sample's axes to match what the next layer reads: a `c(features, seq_len)` tensor into the `c(seq_len, features)` that attention and the recurrent layers expect, or the reverse.

Note that `ggml_layer_embedding` does **not** need this – it already reports `c(seq_len, dim)` and feeds attention, GRU and LSTM directly. Adding a permute there transposes the underlying tensor and breaks the graph.

See Also

`ggml_permute` for the low-level op and its destination-position convention.

Examples

```
# Swap a c(features, seq_len) sample into c(seq_len, features).
model <- ggml_model_sequential() |>
  ggml_layer_permute(c(2, 1), input_shape = c(32, 16))
```

---

ggml_layer_positional_embedding

Add a Learned Positional Embedding

Description

Adds a learned vector to each position of a sequence, so that attention – which is order-blind on its own – can tell one position from another. The table is `[d_model, seq_len]`, one row per position, added to the input and broadcast over the batch.

Usage

```
ggml_layer_positional_embedding(model, name = NULL, trainable = TRUE)
```

Arguments

`model`	A `ggml_sequential_model` or a `ggml_tensor_node` carrying a sequence of shape `c(seq_len, d_model)`.
`name`	Optional character name for the layer.
`trainable`	Logical; whether the table is updated during training.

Details

Unlike ggml_layer_embedding there are no indices to look up: the position IS the place in the sequence, so this layer takes the sequence itself and returns it with the positional term added. The shape is unchanged.

Value

The model with the layer appended, or a new ggml_tensor_node of the same shape as the input.

See Also

ggml_layer_attention, ggml_layer_embedding

Examples

```
# Without this the encoder cannot distinguish "ab" from "ba".
x <- ggml_input(shape = c(10L, 32L))
h <- x |> ggml_layer_positional_embedding()
h <- h |> ggml_layer_attention(d_model = 32L, n_heads = 4L)
```

---

ggml_layer_reshape *Add Reshape Layer*

Description

Reinterpret a sample's shape without moving any data. The batch axis is not part of shape and is never reshaped – shape describes one sample, exactly as input_shape does.

Usage

```
ggml_layer_reshape(
  model,
  shape,
  input_shape = NULL,
  name = NULL,
  trainable = TRUE
)
```

Arguments

model	A ggml_sequential_model object, or a ggml_tensor_node for the functional API.
shape	Integer vector, the new per-sample shape. At most one entry may be -1 to have it inferred. Must not include the batch axis.
input_shape	Input shape, required for the first layer only.
name	Optional character name for the layer.
trainable	Logical; reserved for API consistency (no weights).

Value

The model object with the reshape layer appended (invisibly), or a `ggml_tensor_node` in the functional API.

Inferred axis

One entry of `shape` may be `-1`, in which case it is computed from the element count: `ggml_layer_reshape(c(-1, 32))` on a `c(4, 8, 16)` input gives `c(16, 32)`. At most one axis may be `-1`, and the remaining sizes must divide the element count exactly.

Reshape versus permute

Reshape only relabels the axes; the elements keep their memory order. To actually reorder axes – swapping `c(a, b)` into `c(b, a)`, say – use `ggml_layer_permute`. Using reshape for that silently interleaves the data instead.

Non-contiguous input

`ggml_reshape_*()` requires contiguous data, and a permuted tensor is not contiguous. The layer inserts `ggml_cont` when needed, so it works after a permute; the cost is one copy.

See Also

`ggml_layer_permute` to reorder axes, `ggml_layer_flatten` to collapse them all.

Examples

```
# Split a flat feature vector into a sequence of 32-wide steps.
model <- ggml_model_sequential() |>
  ggml_layer_dense(512, input_shape = 128) |>
  ggml_layer_reshape(c(-1, 32))
```

---

ggml_layer_rms_norm *Add RMS Normalization Layer*

Description

Normalizes each sample by the root mean square of its features, then scales by gamma and shifts by beta (both learnable). Unlike `ggml_layer_batch_norm` it does not subtract the mean and keeps no running statistics, so it behaves identically in training and inference and does not couple the samples in a batch. This is the normalization used by transformer blocks.

Usage

```
ggml_layer_rms_norm(model, eps = 1e-05, name = NULL, trainable = TRUE)
```

Arguments

model	A ggml_sequential_model or a ggml_tensor_node (functional API).
eps	Small constant for numerical stability (default 1e-5)
name	Optional character name for the layer.
trainable	Logical; whether the layer weights are updated during training.

Value

The model with the layer appended, or a new ggml_tensor_node in the functional API.

See Also

ggml_layer_batch_norm

Examples

```
# Functional API: the normalization of a transformer block.
x <- ggml_input(shape = c(10L, 32L))
h <- x |> ggml_layer_rms_norm()
h <- h |> ggml_layer_attention(d_model = 32L, n_heads = 4L)
```

---

ggml_layer_sequence_pooling

Pool a Sequence Down to One Vector

Description

Collapses the sequence axis, turning c(seq_len, d_model) into d_model: the step that lets a classification or regression head sit on top of an encoder. ggml_layer_flatten() also produces a flat vector, but keeps every position separately (seq_len * d_model features), so its width – and the head above it – depends on the sequence length. Pooling does not.

Usage

```
ggml_layer_sequence_pooling(model, mode = c("mean", "first"), name = NULL)
```

Arguments

model	A ggml_sequential_model or a ggml_tensor_node carrying a sequence of shape c(seq_len, d_model).
mode	"mean" (default) or "first".
name	Optional character name for the layer.

Details

`mode = "mean"` averages over the positions; every position contributes equally, which is the usual choice for an encoder without a dedicated summary token. `mode = "first"` takes position 1 and ignores the rest – the CLS-token convention, where attention is expected to have gathered what matters into that position.

Value

The model with the layer appended, or a new `ggml_tensor_node` of shape `d_model`.

See Also

`ggml_layer_flatten`, `ggml_layer_attention`

Examples

```
# An encoder with a regression head: the head's width is d_model, whatever
# the sequence length turns out to be.
x <- ggml_input(shape = c(10L, 32L))
h <- x |> ggml_layer_attention(d_model = 32L, n_heads = 4L)
h <- h |> ggml_layer_sequence_pooling()
y <- h |> ggml_layer_dense(1L)
```

---

ggml_layer_transformer_block

Add a Transformer Encoder Block

Description

The standard pre-LN encoder block, in one call: normalize, attend, add the residual, normalize again, run the position-wise feed-forward, add the second residual. Composed from the existing layers – there is no new node type – so the pieces can still be wired by hand when a block needs to differ.

Usage

```
ggml_layer_transformer_block(
  x,
  d_model,
  n_heads = 1L,
  ff_dim = NULL,
  activation = "silu",
  norm = c("rms", "layer"),
  causal = FALSE,
  mask = NULL,
  rope = FALSE,
```

```
  dropout = 0,
  attn_dropout = dropout,
  name = NULL
)
```

Arguments

x	A ggml_tensor_node of shape c(seq_len, d_model).
d_model	Integer, model width. Must match the input's feature dimension and be divisible by n_heads.
n_heads	Integer, number of attention heads (default 1).
ff_dim	Integer, width of the feed-forward hidden layer. Defaults to 4 * d_model, the usual ratio.
activation	Feed-forward activation (default "silu"). "gelu" is trainable too, so a model ported from a GELU-based configuration keeps its activation. NB: "hardsigmoid" and "hardswish" still have no backward rule in ggml – the graph builds, then training aborts.
norm	"rms" (default) or "layer": which normalization to put in front of each sub-layer.
causal	Logical: causal self-attention (default FALSE).
mask	Optional attention mask node, c(seq_q, seq_kv); see ggml_layer_attention.
rope	Logical: rotary position embedding on queries and keys (default FALSE).
dropout	Dropout rate applied to each sublayer's output before it is added back to the residual (default 0, meaning no dropout).
attn_dropout	Dropout rate for the attention weights specifically; defaults to dropout, so setting only dropout applies the same rate everywhere.
name	Optional prefix for the layers inside the block.

Value

A new ggml_tensor_node of shape c(seq_len, d_model), so blocks stack directly.

Pre-LN vs post-LN

Normalization comes BEFORE each sublayer, not after, which is what makes deep stacks trainable without a warmup schedule: the residual path stays un-normalized from input to output, so the gradient reaches the early layers undiminished. The original 2017 transformer normalized after; every large model since does it this way.

See Also

ggml_layer_attention, ggml_layer_sequence_pooling

Examples

```
# A two-block encoder with a pooled regression head.
x <- ggml_input(shape = c(10L, 32L))
h <- x |> ggml_layer_positional_embedding()
for (i in 1:2) h <- h |> ggml_layer_transformer_block(32L, n_heads = 4L)
y <- h |> ggml_layer_sequence_pooling() |> ggml_layer_dense(1L)
```

---

ggml_leaky_relu *Leaky ReLU Activation (Graph)*

Description

Creates a graph node for Leaky ReLU activation. LeakyReLU(x) = x if x > 0, else negative_slope * x. Unlike standard ReLU, Leaky ReLU allows a small gradient for negative values.

Usage

```
ggml_leaky_relu(ctx, a, negative_slope = 0.01, inplace = FALSE)
```

Arguments

ctx	GGML context
a	Input tensor
negative_slope	Slope for negative values (default: 0.01)
inplace	If TRUE, operation is performed in-place (default: FALSE)

Value

Tensor representing the Leaky ReLU operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
r <- ggml_leaky_relu(ctx, a, negative_slope = 0.1)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # [-0.2, -0.1, 0, 1, 2]
ggml_free(ctx)
```

ggml_load_model *Load a Full Model (Architecture + Weights)*

Description

Restores a model previously saved with `ggml_save_model()`. The returned model is compiled and ready for `ggml_predict()` / `ggml_evaluate()`. Call `ggml_fit()` again to continue training.

Usage

```
ggml_load_model(path, backend = "auto")
```

Arguments

path	File path to an RDS file written by `ggml_save_model()`.
backend	Backend selection: `"auto"`, `"cpu"`, or `"vulkan"`.

Value

A compiled model object.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(16L, activation = "relu", input_shape = 4L) |>
  ggml_layer_dense(2L,  activation = "softmax")
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")
x <- matrix(runif(64 * 4), 64, 4)
y <- matrix(c(rep(c(1,0), 32), rep(c(0,1), 32)), 64, 2)
model <- ggml_fit(model, x, y, epochs = 1L, batch_size = 32L, verbose = 0L)
tmp <- tempfile(fileext = ".rds")
ggml_save_model(model, tmp)
model2 <- ggml_load_model(tmp)
```

ggml_load_weights *Load Model Weights from File*

Description

Loads previously saved weights into a compiled model. The model architecture must match the saved weights (same layer types, sizes, and shapes).

Usage

```
ggml_load_weights(model, path)
```

Arguments

model	A compiled ggml_sequential_model (same architecture as saved)
path	File path to load weights from

Value

The model with loaded weights.

---

ggml_log *Natural Logarithm (Graph)*

Description

Creates a graph node for element-wise natural logarithm: log(x)

Usage

```
ggml_log(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the log operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 3)
ggml_set_f32(a, c(1, exp(1), exp(2)))
result <- ggml_log(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [0, 1, 2]
ggml_free(ctx)
```

ggml_log_inplace *Natural Logarithm In-place (Graph)*

Description

Creates a graph node for in-place element-wise natural logarithm.

Usage

```
ggml_log_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with log values

ggml_log_is_r_enabled *Check if R Logging is Enabled*

Description

Check if R Logging is Enabled

Usage

```
ggml_log_is_r_enabled()
```

Value

Logical indicating if R-compatible logging is active

See Also

Other logging: ggml_abort_is_r_enabled(), ggml_log_set_default(), ggml_log_set_r(), ggml_set_abort_callback_default(), ggml_set_abort_callback_r()

ggml_log_set_default *Restore Default GGML Logging*

Description

Restores GGML to default logging behavior (stderr output).

Usage

```
ggml_log_set_default()
```

Value

NULL invisibly

See Also

Other logging: ggml_abort_is_r_enabled(), ggml_log_is_r_enabled(), ggml_log_set_r(), ggml_set_abort_callback_default(), ggml_set_abort_callback_r()

ggml_log_set_r *Enable R-compatible GGML Logging*

Description

Redirects GGML log messages to R's message system: - INFO/DEBUG messages go to stdout (via Rprintf) - WARN/ERROR messages go to stderr (via REprintf)

Usage

```
ggml_log_set_r()
```

Value

NULL invisibly

See Also

Other logging: ggml_abort_is_r_enabled(), ggml_log_is_r_enabled(), ggml_log_set_default(), ggml_set_abort_callback_default(), ggml_set_abort_callback_r()

Examples

```
ggml_log_set_r()
# Now GGML messages will appear in R console
```

ggml_lstm *Create an LSTM Layer Object*

Description

Create an LSTM Layer Object

Usage

```
ggml_lstm(
  units,
  return_sequences = FALSE,
  activation = "tanh",
  recurrent_activation = "sigmoid",
  name = NULL,
  trainable = TRUE
)
```

Arguments

`units`	Integer, number of hidden units.
`return_sequences`	Logical.
`activation`	Cell gate activation (default `"tanh"`).
`recurrent_activation`	Recurrent gate activation (default `"sigmoid"`).
`name`	Optional character name.
`trainable`	Logical.

Value

A `ggml_layer` object.

ggml_marshal_model *Marshal a ggmlR model to an in-memory container*

Description

Serializes a trained sequential or functional ggmlR model into a self-describing raw container suitable for transport between R sessions or parallel workers (e.g. for **mlr3** parallel resampling and tuning).

Usage

```
ggml_marshal_model(model)
```

Arguments

model	A compiled ggml_sequential_model or ggml_functional_model.

Details

The container wraps the bytes produced by ggml_save_model together with a format tag, schema version, package/R versions, a SHA-256 integrity checksum, and a timestamp. Autograd modules are **not** supported in this version and cause the function to signal an error; the mlr3 learners catch this and fall back to marshaled = FALSE.

Value

A named list with class "ggmlR_marshaled" containing the serialized payload and metadata. Pass it to ggml_unmarshal_model to reconstruct the model.

See Also

ggml_unmarshal_model, ggml_save_model

---

ggml_matmul *GPU matrix multiply (drop-in for %*%)*

Description

Computes A %*% B on the Vulkan GPU and returns an ordinary R matrix, with a transparent CPU fallback. A drop-in accelerator for a large matrix multiply: no autograd, no wrapper types required — plain matrices in and out.

Usage

```
ggml_matmul(A, B, device = "auto", prec = "f32")
```

Arguments

A, B	Numeric matrices with ncol(A) == nrow(B).
device	"auto" (default: GPU when present and the multiply is large enough to amortise the transfer, else CPU), "gpu" (force GPU, still falls back to CPU if none), or "cpu".
prec	"f32" (default; requests f32 accumulation, ~1e-3 relative error or better depending on the driver) or "f16" (faster, never more precise). Only affects the GPU path.

Details

Precision: R multiplies in double precision (f64); a GPU offers f32 at best, so the GPU result never matches R to full double precision. `prec = "f32"` (the default) requests the f32 accumulation kernel, but how close the result actually lands depends on the Vulkan driver: some accumulate `mul_mat` in f16 regardless (e.g. RADV / Mesa), giving a relative error around `1e-3` either way. Treat the GPU path as a fast, approximate multiply — typically `1e-3`, better on drivers with true f32 accumulation — not a bit-for-bit replacement for `%*%`. `prec = "f16"` only ever lowers precision; use it when speed matters more than the last digits.

Value

The product `A %*% B` as a numeric `matrix`.

See Also

`ggml_crossprod`, `ggml_tcrossprod`, `as_gpu_matrix`

Examples

```
A <- matrix(rnorm(4), 2); B <- matrix(rnorm(4), 2)
ggml_matmul(A, B, device = "cpu")
```

---

ggml_matmul_f64 *GPU double-precision matrix multiply*

Description

Computes `A %*% B` on the Vulkan GPU in **full double precision** (fp64), returning an ordinary R matrix, with a transparent CPU fallback. Unlike `ggml_matmul` (which is f32/f16 and approximate), this matches R's `%*%` to machine precision (`~1e-15`), because the kernel accumulates in `double` throughout.

Usage

```
ggml_matmul_f64(A, B, device = "auto")
```

Arguments

`A, B`	Numeric matrices with `ncol(A) == nrow(B)`.
`device`	`"auto"` (default: GPU when present, fp64-capable and the multiply is large enough to amortise the transfer, else CPU), `"gpu"` (force GPU, still falls back to CPU if fp64 is unavailable), or `"cpu"`.

Details

fp64 on the GPU is only worthwhile on hardware with fast double throughput: data-centre cards (NVIDIA Tesla P100/V100, AMD Instinct) run fp64 at 1:2 of fp32, whereas consumer GPUs cripple it (RDNA ~1:16, GeForce ~1:64), where the CPU (multithreaded double BLAS) is usually faster. Use this when you need bit-accurate double results on a capable GPU, or for numerically sensitive work where the f32 path's error is unacceptable; otherwise prefer `ggml_matmul`. Requires the `shaderFloat64` device feature — without it (or without a GPU), the multiply silently falls back to the CPU.

Value

The product `A %*% B` as a numeric `matrix`, accurate to double precision.

See Also

`ggml_matmul` (faster f32/f16 approximate path).

Examples

```
A <- matrix(rnorm(4), 2); B <- matrix(rnorm(4), 2)
ggml_matmul_f64(A, B, device = "cpu")
```

---

ggml_matrix-class *A GPU-backed matrix wrapper*

Description

An S4 class holding an ordinary numeric matrix, tagged so that `%*%`, `crossprod()` and `tcrossprod()` run on the GPU (see `ggml_matmul`). Wrap a matrix with `as_gpu_matrix`.

Usage

```
## S4 method for signature 'ggml_matrix,ANY'
x %*% y

## S4 method for signature 'ANY,ggml_matrix'
x %*% y

## S4 method for signature 'ggml_matrix,ggml_matrix'
x %*% y

## S4 method for signature 'ggml_matrix'
crossprod(x, y)

## S4 method for signature 'ggml_matrix'
tcrossprod(x, y)
```

```
## S4 method for signature 'ggml_matrix'
dim(x)

## S4 method for signature 'ggml_matrix'
show(object)
```

Arguments

x, y	Operands; at least one is a ggml_matrix (the other is coerced with as.matrix). For crossprod/tcrossprod, y may be missing.
object	A ggml_matrix (for the show method).

Value

%*%, crossprod and tcrossprod return a plain numeric matrix (the GPU result); dim an integer vector.

Slots

data The underlying numeric matrix.

device Dispatch preference: "auto", "gpu" or "cpu".

prec Precision of the GPU path: "f32" or "f16".

---

ggml_mean *Mean (Graph)*

Description

Creates a graph node that computes the mean of all elements.

Usage

```
ggml_mean(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Scalar tensor with the mean

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(2, 4, 6, 8, 10))
result <- ggml_mean(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # 6
ggml_free(ctx)
```

---

ggml_model *Create a Functional Model*

Description

Assembles a ggml_functional_model from symbolic input and output nodes produced by ggml_input() and ggml_layer_*() calls.

Usage

```
ggml_model(inputs, outputs)
```

Arguments

inputs	A ggml_tensor_node or a list of them (model inputs).
outputs	A ggml_tensor_node or a list of them (model outputs).

Value

A ggml_functional_model object.

Examples

```
x   <- ggml_input(shape = 64L)
out <- x |> ggml_layer_dense(10, activation = "softmax")
model <- ggml_model(inputs = x, outputs = out)
```

ggml_model_backend *Backend a fitted ggml model actually ran on*

Description

Reports the backend the model was *actually* compiled onto, making a silent `backend = "auto"` fallback to CPU (when no GPU is available) inspectable. Works on a raw sequential/functional model or a fitted parsnip engine object.

Usage

```
ggml_model_backend(object, verbose = FALSE)
```

Arguments

`object`	A compiled/fitted `ggml_sequential_model`, `ggml_functional_model` or `ggmlr_parsnip_model`.
`verbose`	If `FALSE` (default) returns a single string (`"vulkan"` or `"cpu"`). If `TRUE` returns a list with details (see Value).

Value

If `verbose = FALSE`, a length-1 character: the backend in use (`"vulkan"` or `"cpu"`). If `verbose = TRUE`, a list with: `requested` (what was asked: `"auto"`/`"cpu"`/`"vulkan"`), `used` (`"vulkan"`/`"cpu"`), `device` (GPU device description, or `"cpu"`) and `fallback` (logical: `TRUE` when a non-CPU backend was requested but CPU was used instead).

See Also

`ggml_training_history`

ggml_model_sequential *Create a Sequential Neural Network Model*

Description

Creates an empty sequential model that layers can be added to using pipe (`|>`) operators.

Usage

```
ggml_model_sequential()
```

Value

A ggml_sequential_model object

Examples

```
## Not run:
model <- ggml_model_sequential() |>
  ggml_layer_conv_2d(32, c(3,3), activation = "relu",
                     input_shape = c(28, 28, 1)) |>
  ggml_layer_max_pooling_2d(c(2,2)) |>
  ggml_layer_flatten() |>
  ggml_layer_dense(128, activation = "relu") |>
  ggml_layer_dense(10, activation = "softmax")

## End(Not run)
```

---

ggml_mul *Multiply tensors*

Description

Creates a graph node for element-wise multiplication.

Usage

```
ggml_mul(ctx, a, b)

ggml_mul(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor
b	Second tensor (same shape as a)

Value

Tensor representing the multiplication operation

Tensor representing the multiplication operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(2, 2, 2, 2, 2))
result <- ggml_mul(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
ggml_set_f32(b, c(2, 2, 2, 2, 2))
result <- ggml_mul(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

---

ggml_mul_inplace *Element-wise Multiplication In-place (Graph)*

Description

Creates a graph node for in-place element-wise multiplication. Result is stored in tensor a, saving memory allocation.

Usage

```
ggml_mul_inplace(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor (will be modified in-place)
b	Second tensor (same shape as a)

Value

View of tensor a with the multiplication result

---

ggml_mul_mat *Matrix Multiplication (Graph)*

Description

Creates a graph node for matrix multiplication. CRITICAL for LLM operations. For matrices A (m x n) and B (n x p), computes C = A * B (m x p).

Usage

```
ggml_mul_mat(ctx, a, b)
```

Arguments

ctx	GGML context
a	First matrix tensor
b	Second matrix tensor

Value

Tensor representing the matrix multiplication

Examples

```
ctx <- ggml_init(1024 * 1024)
A <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 3)
B <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 4, 2)
ggml_set_f32(A, 1:12)
ggml_set_f32(B, 1:8)
C <- ggml_mul_mat(ctx, A, B)
graph <- ggml_build_forward_expand(ctx, C)
ggml_graph_compute(ctx, graph)
ggml_get_f32(C)
ggml_free(ctx)
```

---

ggml_mul_mat_id *Matrix Multiplication with Expert Selection (Graph)*

Description

Indirect matrix multiplication for Mixture of Experts architectures. Selects expert weights based on indices and performs batched matmul.

Usage

```
ggml_mul_mat_id(ctx, as, b, ids)
```

Arguments

ctx	GGML context
as	Stacked expert weight matrices [n_embd, n_ff, n_experts]
b	Input tensor
ids	Expert selection indices tensor (I32)

Value

Output tensor after expert-selected matrix multiplication

Examples

```
ctx <- ggml_init(64 * 1024 * 1024)
# 4 experts, each with 8x16 weights (small for example)
experts <- ggml_new_tensor_3d(ctx, GGML_TYPE_F32, 8, 16, 4)
ggml_set_f32(experts, rnorm(8 * 16 * 4))
input <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 8, 2)
ggml_set_f32(input, rnorm(16))
# Select expert 0 for token 0, expert 2 for token 1
ids <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 2)
ggml_set_i32(ids, c(0L, 2L))
output <- ggml_mul_mat_id(ctx, experts, input, ids)
graph <- ggml_build_forward_expand(ctx, output)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_n_dims *Get Number of Dimensions*

Description

Returns the number of dimensions of a tensor

Usage

```
ggml_n_dims(tensor)
```

Arguments

tensor	Tensor pointer

Value

Integer number of dimensions (1-4)

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_n_dims(t)
ggml_free(ctx)
```

GGML_N_TASKS_MAX *Maximum Task Count for Custom Ops*

Description

Sentinel for the `n_tasks` argument of `ggml_custom` and `ggml_custom_inplace`, meaning "use as many threads as the backend allows" (mirrors `GGML_N_TASKS_MAX` in ggml.h).

Usage

```
GGML_N_TASKS_MAX
```

Format

An object of class `integer` of length 1.

ggml_nbytes *Get number of bytes*

Description

Get number of bytes

Get Number of Bytes

Usage

```
ggml_nbytes(tensor)

ggml_nbytes(tensor)
```

Arguments

`tensor`	Tensor

Value

Integer number of bytes

Integer number of bytes

Examples

```
ctx <- ggml_init(1024 * 1024)
tensor <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_nbytes(tensor)
ggml_free(ctx)


ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_nbytes(t)
ggml_free(ctx)
```

---

ggml_neg *Negation (Graph)*

Description

Creates a graph node for element-wise negation: -x

Usage

```
ggml_neg(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the negation operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, -2, 3, -4))
result <- ggml_neg(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [-1, 2, -3, 4]
ggml_free(ctx)
```

ggml_neg_inplace *Negation In-place (Graph)*

Description

Creates a graph node for in-place element-wise negation: -x

Usage

```
ggml_neg_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with negated values

ggml_nelements *Get number of elements*

Description

Get number of elements

Get Number of Elements

Usage

```
ggml_nelements(tensor)

ggml_nelements(tensor)
```

Arguments

tensor	Tensor

Value

Integer number of elements

Integer number of elements

Examples

```
ctx <- ggml_init(1024 * 1024)
tensor <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_nelements(tensor)
ggml_free(ctx)


ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_nelements(t)
ggml_free(ctx)
```

---

ggml_new_f32 *Create Scalar F32 Tensor*

Description

Creates a 1-element tensor containing a single float value. Useful for scalar operations, learning rates, and other scalar floats.

Usage

```
ggml_new_f32(ctx, value)
```

Arguments

ctx	GGML context
value	Numeric value

Value

Tensor pointer (1-element F32 tensor)

Examples

```
ctx <- ggml_init(1024 * 1024)
scalar <- ggml_new_f32(ctx, 3.14)
ggml_get_f32(scalar)
ggml_free(ctx)
```

ggml_new_i32 *Create Scalar I32 Tensor*

Description

Creates a 1-element tensor containing a single integer value. Useful for indices, counters, and other scalar integer operations.

Usage

```
ggml_new_i32(ctx, value)
```

Arguments

ctx	GGML context
value	Integer value

Value

Tensor pointer (1-element I32 tensor)

Examples

```
ctx <- ggml_init(1024 * 1024)
scalar <- ggml_new_i32(ctx, 42)
ggml_get_i32(scalar)
ggml_free(ctx)
```

ggml_new_tensor *Create Tensor with Arbitrary Dimensions*

Description

Generic tensor constructor for creating tensors with 1-4 dimensions. This is more flexible than the ggml_new_tensor_Nd functions.

Usage

```
ggml_new_tensor(ctx, type = GGML_TYPE_F32, n_dims, ne)
```

Arguments

ctx	GGML context
type	Data type (GGML_TYPE_F32, etc.)
n_dims	Number of dimensions (1-4)
ne	Numeric vector of dimension sizes

Value

Tensor pointer

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor(ctx, GGML_TYPE_F32, 3, c(10, 20, 30))
ggml_nelements(t)
ggml_free(ctx)
```

---

ggml_new_tensor_1d *Create 1D tensor*

Description

Create 1D tensor

Create 1D Tensor

Usage

```
ggml_new_tensor_1d(ctx, type = GGML_TYPE_F32, ne0)

ggml_new_tensor_1d(ctx, type = GGML_TYPE_F32, ne0)
```

Arguments

ctx	GGML context
type	Data type
ne0	Size

Value

Tensor pointer

Tensor pointer

Examples

```
ctx <- ggml_init(1024 * 1024)
tensor <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_nelements(tensor)
ggml_free(ctx)
```

ggml_new_tensor_2d *Create 2D tensor*

Description

Create 2D tensor

Create 2D Tensor

Usage

```
ggml_new_tensor_2d(ctx, type = GGML_TYPE_F32, ne0, ne1)

ggml_new_tensor_2d(ctx, type = GGML_TYPE_F32, ne0, ne1)
```

Arguments

ctx	GGML context
type	Data type
ne0	Rows
ne1	Columns

Value

Tensor pointer

Tensor pointer

Examples

```
ctx <- ggml_init(1024 * 1024)
tensor <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_nelements(tensor)
ggml_free(ctx)
```

ggml_new_tensor_3d *Create 3D Tensor*

Description

Create 3D Tensor

Usage

```
ggml_new_tensor_3d(ctx, type = GGML_TYPE_F32, ne0, ne1, ne2)
```

Arguments

ctx	GGML context
type	Data type (default GGML_TYPE_F32)
ne0	Size of dimension 0
ne1	Size of dimension 1
ne2	Size of dimension 2

Value

Tensor pointer

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_3d(ctx, GGML_TYPE_F32, 10, 20, 30)
ggml_nelements(t)
ggml_free(ctx)
```

---

ggml_new_tensor_4d *Create 4D Tensor*

Description

Create 4D Tensor

Usage

```
ggml_new_tensor_4d(ctx, type = GGML_TYPE_F32, ne0, ne1, ne2, ne3)
```

Arguments

ctx	GGML context
type	Data type (default GGML_TYPE_F32)
ne0	Size of dimension 0
ne1	Size of dimension 1
ne2	Size of dimension 2
ne3	Size of dimension 3

Value

Tensor pointer

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, 8, 8, 3, 2)
ggml_nelements(t)
ggml_free(ctx)
```

`ggml_norm` *Layer Normalization (Graph)*

Description

Creates a graph node for layer normalization. Normalizes input to zero mean and unit variance.

Usage

```
ggml_norm(ctx, a, eps = 1e-05)
```

Arguments

`ctx`	GGML context
`a`	Input tensor
`eps`	Epsilon value for numerical stability (default: 1e-5)

Value

Tensor representing the layer normalization operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_norm(ctx, a, eps = 1e-5)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)  # Normalized values
ggml_free(ctx)
```

ggml_norm_back *Layer Norm Backward (Graph)*

Description

Creates a graph node for the backward pass of ggml_norm, i.e. of LayerNorm without gamma/beta. Upstream ggml has no backward for GGML_OP_NORM; ggmlR adds it with CPU and Vulkan kernels, which is what makes ggml_layer_layer_norm trainable.

Usage

```
ggml_norm_back(ctx, a, b, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Gradient tensor (dy from the forward output)
b	Input tensor (x from the forward pass)
eps	Epsilon for numerical stability (default 1e-5)

Value

Tensor representing the gradient with respect to input

See Also

ggml_norm, ggml_rms_norm_back

ggml_norm_inplace *Layer Normalization In-place (Graph)*

Description

Creates a graph node for in-place layer normalization. Returns a view of the input tensor.

Usage

```
ggml_norm_inplace(ctx, a, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)
eps	Epsilon value for numerical stability (default: 1e-5)

Value

View of input tensor with layer normalization applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_norm_inplace(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_nrows *Get Number of Rows*

Description

Returns the number of rows in a tensor (product of all dimensions except ne[0]).

Usage

```
ggml_nrows(tensor)
```

Arguments

tensor	Tensor pointer

Value

Number of rows

---

ggml_op_can_inplace *Check if Operation Can Be Done In-place*

Description

Returns whether a GGML operation can reuse memory from its source tensors. This is useful for memory optimization.

Usage

```
ggml_op_can_inplace(op)
```

Arguments

op	Operation code (integer)

Value

Logical indicating if operation supports in-place execution

See Also

Other graph: ggml_graph_view()

Examples

```
# Check if operation code 1 (ADD) can be in-place
can_inplace <- ggml_op_can_inplace(1L)
```

---

ggml_op_desc *Get Operation Description from Tensor*

Description

Returns a description of the operation that produces a tensor.

Usage

```
ggml_op_desc(tensor)
```

Arguments

tensor	Tensor pointer

Value

Character string describing the operation

See Also

Other op_info: ggml_get_unary_op(), ggml_op_name(), ggml_op_symbol(), ggml_unary_op_name()

ggml_op_name *Get Operation Name*

Description

Returns the string name of a GGML operation.

Usage

```
ggml_op_name(op)
```

Arguments

op	GGML operation constant

Value

Character string with operation name

See Also

Other op_info: ggml_get_unary_op(), ggml_op_desc(), ggml_op_symbol(), ggml_unary_op_name()

ggml_op_symbol *Get Operation Symbol*

Description

Returns the mathematical symbol for a GGML operation.

Usage

```
ggml_op_symbol(op)
```

Arguments

op	GGML operation constant

Value

Character string with operation symbol

See Also

Other op_info: ggml_get_unary_op(), ggml_op_desc(), ggml_op_name(), ggml_unary_op_name()

ggml_ops_registry *Supported single-cell operations*

Description

Returns the registry of operations the single-cell adapter can dispatch. Use this to check capabilities (and required parameters) before building a ggml_task — capability is declared, never discovered at runtime.

Usage

```
ggml_ops_registry(op = NULL)
```

Arguments

op	Optional operation name. If supplied, returns that single entry (or NULL if unknown); otherwise a named list of all entries.

Value

A list describing the operation(s): op, params (required parameter names) and desc.

Examples

```
ggml_ops_registry()
ggml_ops_registry("embed")
```

ggml_opt_alloc *Allocate graph for evaluation*

Description

Must be called before ggml_opt_eval. Allocates forward or forward+backward graph.

Usage

```
ggml_opt_alloc(opt_ctx, backward = TRUE)
```

Arguments

opt_ctx	External pointer to optimizer context
backward	Whether to allocate backward graph (for training)

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_context_optimizer_ty
ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_ba
ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(
ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(),
ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(),
ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(),
ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(),
ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(),
ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh
ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(),
ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(),
ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(),
ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(),
ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(),
ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_context_optimizer_type

Get optimizer type from context

Description

Get optimizer type from context

Usage

```
ggml_opt_context_optimizer_type(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

Integer optimizer type constant

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(),
ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_ba
ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(
ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(),
ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(),
ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(),
ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(),
ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(),

ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh
ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(),
ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(),
ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(),
ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(),
ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(),
ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_dataset_data` *Get data tensor from dataset*

Description

Returns the underlying data tensor with shape [ne_datapoint, ndata].

Usage

```
ggml_opt_dataset_data(dataset)
```

Arguments

`dataset`	External pointer to dataset

Value

External pointer to data tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(),
ggml_opt_context_optimizer_type(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(),
ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(),
ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(),
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(),
ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(),
ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(),
ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(),
ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse
ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(),
ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(),
ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(),
ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(),
ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(),
ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(),
ggml_snapshot_free(), ggml_trainer()

ggml_opt_dataset_free *Free optimization dataset*

Description

Releases memory associated with a dataset.

Usage

```
ggml_opt_dataset_free(dataset)
```

Arguments

dataset	External pointer to dataset

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_dataset_get_batch *Get batch from dataset*

Description

Copies a batch of data and labels to the provided tensors.

Usage

```
ggml_opt_dataset_get_batch(dataset, data_batch, labels_batch = NULL, ibatch)
```

Arguments

dataset	External pointer to dataset
data_batch	Tensor to receive data batch
labels_batch	Tensor to receive labels batch (can be NULL)
ibatch	Batch index

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_dataset_get_batch_head

Get one output head's slice of a batch's labels

Description

Multi-output counterpart of ggml_opt_dataset_get_batch(). The dataset's labels hold every head's labels concatenated along the first dimension; this copies just the slice belonging to one head.

Usage

```
ggml_opt_dataset_get_batch_head(
  dataset,
  data_batch,
  labels_batch,
  labels_off,
  ibatch
)
```

Arguments

dataset	Dataset pointer
data_batch	Input batch tensor, or NULL to copy only the labels (the caller copies the inputs once, with the first head)
labels_batch	This head's labels batch tensor
labels_off	Offset of this head within a label row, 0-based, in elements
ibatch	Batch index, 0-based

Value

NULL, invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse, ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_dataset_init` *Create a new optimization dataset*

Description

Creates a dataset for training with specified data and label types.

Usage

```
ggml_opt_dataset_init(
  type_data,
  type_label,
  ne_datapoint,
  ne_label,
  ndata,
  ndata_shard = 1
)
```

Arguments

`type_data`	GGML type for data tensor (e.g., GGML_TYPE_F32)
`type_label`	GGML type for label tensor (e.g., GGML_TYPE_F32)
`ne_datapoint`	Number of elements per datapoint
`ne_label`	Number of elements per label (0 if no labels)
`ndata`	Total number of datapoints
`ndata_shard`	Shard size for shuffling (1 is fine for most cases)

Value

External pointer to dataset

See Also

Other optimization: `ggml_fit_opt()`, `ggml_fit_opt_gen()`, `ggml_fit_opt_multi()`, `ggml_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `ggml_opt_dataset_get_batch_head()`, `ggml_opt_dataset_labels()`, `ggml_opt_dataset_ndata()`, `ggml_opt_dataset_shuffle()`, `ggml_opt_dataset_weights()`, `ggml_opt_default_params()`, `ggml_opt_epoch()`, `ggml_opt_eval()`, `ggml_opt_fit()`, `ggml_opt_fit_multi()`, `ggml_opt_free()`, `ggml_opt_get_lr()`, `ggml_opt_grad_acc()`, `ggml_opt_init()`, `ggml_opt_init_for_fit()`, `ggml_opt_init_for_fit_multi()`, `ggml_opt_inputs()`, `ggml_opt_labels()`, `ggml_opt_loss()`, `ggml_opt_loss_i()`, `ggml_opt_loss_type_binary_cross_entropy()`, `ggml_opt_loss_type_cross_entropy()`, `ggml_opt_loss_type_huber()`, `ggml_opt_loss_type_mae()`, `ggml_opt_loss_type_mean()`, `ggml_opt_loss_type_mse`, `ggml_opt_loss_type_sum()`, `ggml_opt_loss_type_weighted_mse()`, `ggml_opt_n_loss()`, `ggml_opt_ncorrect()`, `ggml_opt_optimizer_name()`, `ggml_opt_optimizer_type_adamw()`, `ggml_opt_optimizer_type_sgd()`, `ggml_opt_outputs()`, `ggml_opt_outputs_i()`, `ggml_opt_pred()`, `ggml_opt_prepare_alloc()`, `ggml_opt_reset()`, `ggml_opt_result_accuracy()`, `ggml_opt_result_free()`, `ggml_opt_result_init()`,

ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_dataset_labels` *Get labels tensor from dataset*

Description

Returns the underlying labels tensor with shape [ne_label, ndata].

Usage

```
ggml_opt_dataset_labels(dataset)
```

Arguments

`dataset`	External pointer to dataset

Value

External pointer to labels tensor, or NULL if no labels

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse, ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_dataset_ndata *Get number of datapoints in dataset*

Description

Get number of datapoints in dataset

Usage

```
ggml_opt_dataset_ndata(dataset)
```

Arguments

dataset	External pointer to dataset

Value

Number of datapoints

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(), ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse, ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_dataset_shuffle *Shuffle dataset*

Description

Shuffles the dataset using the RNG from the optimizer context.

Usage

```
ggml_opt_dataset_shuffle(opt_ctx, dataset, idata = -1)
```

Arguments

opt_ctx	External pointer to optimizer context
dataset	External pointer to dataset
idata	Number of datapoints to shuffle (-1 for all)

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_weights(), ggml_opt_default_paran ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_ ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_dataset_weights` *Get dataset loss weights tensor*

Description

Returns the (lazily allocated) loss weights tensor with shape `[ne0, ndata]`. The first call allocates it; fill it via `ggml_backend_tensor_set_data()`. Used together with `ggml_opt_loss_type_weighted_mse`.

Usage

```
ggml_opt_dataset_weights(dataset, ne0 = 1L)
```

Arguments

`dataset`	External pointer to dataset
`ne0`	Width of the weight tensor: 1 for per-datapoint weights, or the label width for a per-output mask.

Details

`ne0 = 1` gives one weight per datapoint, broadcast over the outputs – the original behaviour. `ne0 = ne_label` gives a per-output-element mask, which is what a Q-model needs: weight 1 on the action that has a TD target, 0 elsewhere, so only that coordinate is trained.

The mask does not change the denominator: the loss stays `sum(w * (pred - y)^2) / nelements(pred)`, so zeroed elements drop out of the numerator without renormalising over the active coordinates.

Calling this again with a different `ne0` than the allocation is an error rather than a silent return of the cached tensor – both shapes broadcast, so a mismatch would otherwise weight the wrong thing unnoticed.

Value

External pointer to weights tensor

See Also

Other optimization: `ggml_fit_opt()`, `ggml_fit_opt_gen()`, `ggml_fit_opt_multi()`, `ggml_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `ggml_opt_dataset_get_batch_head()`, `ggml_opt_dataset_init()`, `ggml_opt_dataset_labels()`, `ggml_opt_dataset_ndata()`, `ggml_opt_dataset_shuffle()`, `ggml_opt_default_paran` `ggml_opt_epoch()`, `ggml_opt_eval()`, `ggml_opt_fit()`, `ggml_opt_fit_multi()`, `ggml_opt_free()`, `ggml_opt_get_lr()`, `ggml_opt_grad_acc()`, `ggml_opt_init()`, `ggml_opt_init_for_fit()`, `ggml_opt_init_for_fit_` `ggml_opt_inputs()`, `ggml_opt_labels()`, `ggml_opt_loss()`, `ggml_opt_loss_i()`, `ggml_opt_loss_type_binary_cros` `ggml_opt_loss_type_cross_entropy()`, `ggml_opt_loss_type_huber()`, `ggml_opt_loss_type_mae()`, `ggml_opt_loss_type_mean()`, `ggml_opt_loss_type_mse()`, `ggml_opt_loss_type_sum()`, `ggml_opt_loss_type_weigh` `ggml_opt_n_loss()`, `ggml_opt_ncorrect()`, `ggml_opt_optimizer_name()`, `ggml_opt_optimizer_type_adamw()`,

ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_default_params` *Get default optimizer parameters*

Description

Returns a list with default optimization parameters.

Usage

```
ggml_opt_default_params(sched, loss_type)
```

Arguments

sched	Backend scheduler
loss_type	Loss type constant

Value

List with loss_type, build_type, opt_period, optimizer

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_ ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_epoch *Run one training epoch*

Description

Performs training on the front portion of the dataset and evaluation on the back portion. This gives more control than ggml_opt_fit.

Usage

```
ggml_opt_epoch(
  opt_ctx,
  dataset,
  result_train = NULL,
  result_eval = NULL,
  idata_split,
  callback_train = TRUE,
  callback_eval = TRUE
)
```

Arguments

opt_ctx	External pointer to optimizer context
dataset	External pointer to dataset
result_train	Result object to accumulate training stats (or NULL)
result_eval	Result object to accumulate evaluation stats (or NULL)
idata_split	Data index at which to split training and evaluation
callback_train	Callback for training: TRUE for progress bar, FALSE for none, or a function(train, ibatch, ibatch_max, t_start_us, result)
callback_eval	Callback for evaluation: TRUE for progress bar, FALSE for none, or a function(train, ibatch, ibatch_max, t_start_us, result)

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_ ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros

ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

Examples

```
# Requires full optimizer setup - see ggml_opt_fit() for simpler API
if (FALSE) {
result_train <- ggml_opt_result_init()
result_eval <- ggml_opt_result_init()
ggml_opt_epoch(opt_ctx, dataset, result_train, result_eval,
               idata_split = 900, callback_train = TRUE)
ggml_opt_result_free(result_train)
ggml_opt_result_free(result_eval)
}
```

---

ggml_opt_eval *Evaluate model*

Description

Performs forward pass, optionally increments result, and does backward pass if allocated.

Usage

```
ggml_opt_eval(opt_ctx, result = NULL)
```

Arguments

opt_ctx	External pointer to optimizer context
result	External pointer to result object (optional)

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_fit(), ggml_opt_fit_multi(),

ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_fit` *Fit model to dataset*

Description

High-level function to train a model on a dataset. This is the recommended way to train models.

Usage

```
ggml_opt_fit(
  sched,
  ctx_compute,
  inputs,
  outputs,
  dataset,
  loss_type = ggml_opt_loss_type_mse(),
  optimizer = ggml_opt_optimizer_type_adamw(),
  nepoch = 1,
  nbatch_logical = 32,
  val_split = 0,
  silent = FALSE
)
```

Arguments

`sched`	Backend scheduler
`ctx_compute`	Compute context (for temporary tensors)
`inputs`	Input tensor with shape [ne_datapoint, batch_size]
`outputs`	Output tensor with shape [ne_label, batch_size]
`dataset`	Dataset created with ggml_opt_dataset_init
`loss_type`	Loss type (default: MSE)
`optimizer`	Optimizer type (default: AdamW)
`nepoch`	Number of epochs

nbatch_logical	Logical batch size (for gradient accumulation)
val_split	Fraction of data for validation (0.0 to 1.0)
silent	Whether to suppress progress output

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

Examples

```
# Full training requires building a computation graph
# See package vignettes for complete examples
if (FALSE) {
cpu <- ggml_backend_cpu_init()
sched <- ggml_backend_sched_new(list(cpu))
dataset <- ggml_opt_dataset_init(GGML_TYPE_F32, GGML_TYPE_F32, 10, 1, 1000)
# ... build model graph with ctx_compute, inputs, outputs ...
ggml_opt_fit(sched, ctx_compute, inputs, outputs, dataset,
             nepoch = 10, val_split = 0.1)
ggml_opt_dataset_free(dataset)
ggml_backend_sched_free(sched)
ggml_backend_free(cpu)
}
```

---

ggml_opt_fit_multi *Fit a multi-output model*

Description

Trains a model with several output heads. Each head has its own labels, loss type and weight; the optimized total is `sum(loss_weights[i] * loss_i)`.

Usage

```
ggml_opt_fit_multi(
  sched,
  ctx_compute,
  inputs,
  outputs,
  dataset,
  loss_types,
  loss_weights = rep(1, length(outputs)),
  labels_offs = NULL,
  optimizer = ggml_opt_optimizer_type_adamw(),
  nepoch = 1,
  nbatch_logical = 32,
  val_split = 0,
  silent = FALSE
)
```

Arguments

`sched`	Backend scheduler
`ctx_compute`	Compute context holding the model graph
`inputs`	Input tensor
`outputs`	List of output tensors, one per head
`dataset`	Dataset with data and concatenated labels
`loss_types`	Integer vector of loss type constants, one per head
`loss_weights`	Numeric vector of head weights (default: all 1)
`labels_offs`	Numeric vector of label offsets, 0-based, one per head
`optimizer`	Optimizer type constant
`nepoch`	Number of epochs
`nbatch_logical`	Logical batch size
`val_split`	Fraction of the data used for validation
`silent`	Suppress progress output

Details

All heads share the dataset's datapoints. The dataset's labels hold every head's labels concatenated along the first dimension: head `i` occupies `labels_offs[i] + seq_len(width_i)` of each label row.

Value

Named list with train_loss, train_accuracy, val_loss, val_accuracy, head_loss (a nepoch x nhead matrix of unweighted per-head losses) and head_accuracy (same shape; NA for heads whose loss is not cross-entropy)

See Also

ggml_fit_opt_multi, which this wraps and which adds callbacks and shuffling control

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_ ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_free *Free optimizer context*

Description

Releases memory associated with an optimizer context.

Usage

```
ggml_opt_free(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_ ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_get_lr *Get current learning rate from optimizer context*

Description

Get current learning rate from optimizer context

Usage

```
ggml_opt_get_lr(lr_ud)
```

Arguments

lr_ud	LR userdata pointer (from 'ggml_opt_init_for_fit()$lr_ud')

Value

Named numeric vector with 'adamw' and 'sgd' LR values

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_mu ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh

ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_grad_acc` *Get gradient accumulator for a tensor*

Description

Returns the gradient accumulator tensor for a node from the forward graph.

Usage

```
ggml_opt_grad_acc(opt_ctx, node)
```

Arguments

`opt_ctx`	External pointer to optimizer context
`node`	External pointer to tensor node

Value

External pointer to gradient accumulator tensor, or NULL if not found

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_mult
ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros
ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh
ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_init *Initialize optimizer context*

Description

Creates a new optimizer context for training.

Usage

```
ggml_opt_init(
  sched,
  loss_type,
  optimizer = ggml_opt_optimizer_type_adamw(),
  opt_period = 1L,
  ctx_compute = NULL,
  inputs = NULL,
  outputs = NULL
)
```

Arguments

sched	Backend scheduler
loss_type	Loss type (use ggml_opt_loss_type_* functions)
optimizer	Optimizer type (use ggml_opt_optimizer_type_* functions)
opt_period	Gradient accumulation steps before optimizer step
ctx_compute	Compute context for static graph mode (or NULL)
inputs	Input tensor for static graph mode (or NULL)
outputs	Output tensor for static graph mode (or NULL)

Value

External pointer to optimizer context

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_ ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(),

ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_init_for_fit` *Initialize optimizer context for R-side epoch loop*

Description

Returns a list with 'opt_ctx' and 'lr_ud' (learning rate userdata pointer). Use 'ggml_opt_set_lr()' to update LR between epochs. The optimizer state (momentum) is preserved across epochs.

Usage

```
ggml_opt_init_for_fit(
  sched,
  loss_type,
  optimizer = ggml_opt_optimizer_type_adamw(),
  opt_period = 1L,
  ctx_compute = NULL,
  inputs = NULL,
  outputs = NULL,
  loss_mask_ne0 = NULL,
  weight_decay = 0
)
```

Arguments

`sched`	Backend scheduler
`loss_type`	Loss type constant
`optimizer`	Optimizer type constant
`opt_period`	Gradient accumulation period
`ctx_compute`	Compute context (for static graphs)
`inputs`	Input tensor (for static graphs)
`outputs`	Output tensor (for static graphs)
`loss_mask_ne0`	Width of the weighted-MSE weight tensor: `NULL` or 1 for per-datapoint weights, or the output width for a per-output mask. Ignored by every other loss type.
`weight_decay`	Decoupled weight decay, in `[0, 1]`; 0 turns it off. Applied to trainable parameters only, so frozen weights are not decayed.

Value

List with elements 'opt_ctx' and 'lr_ud'

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_init_for_fit_multi

Initialize a multi-output optimizer context for an R-side epoch loop

---

Description

Multi-output counterpart of ggml_opt_init_for_fit(). Returns the same opt_ctx/lr_ud pair, so ggml_opt_set_lr() and hence LR schedulers work identically; in addition it records each head's loss type, weight and label offset, which ggml_opt_fit_multi() would otherwise set internally.

Usage

```
ggml_opt_init_for_fit_multi(
  sched,
  loss_types,
  loss_weights,
  labels_offs,
  optimizer = ggml_opt_optimizer_type_adamw(),
  opt_period = 1L,
  ctx_compute = NULL,
  inputs = NULL,
  outputs = NULL,
  weight_decay = 0
)
```

Arguments

sched	Backend scheduler
loss_types	Integer vector of loss type constants, one per head

loss_weights	Numeric vector of head weights, one per head
labels_offs	Numeric vector of 0-based label offsets, one per head
optimizer	Optimizer type constant
opt_period	Gradient accumulation period
ctx_compute	Compute context holding the model graph
inputs	Input tensor
outputs	List of output tensors, one per head
weight_decay	As for ggml_opt_init_for_fit().

Value

List with elements 'opt_ctx' and 'lr_ud'

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cros ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weigh ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_inputs *Get inputs tensor from optimizer context*

Description

Get inputs tensor from optimizer context

Usage

```
ggml_opt_inputs(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

External pointer to inputs tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_hub ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum() ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_labels *Get labels tensor from optimizer context*

Description

Get labels tensor from optimizer context

Usage

```
ggml_opt_labels(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

External pointer to labels tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh

ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_hube ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum() ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_loss` *Get loss tensor from optimizer context*

Description

Get loss tensor from optimizer context

Usage

```
ggml_opt_loss(opt_ctx)
```

Arguments

`opt_ctx`	External pointer to optimizer context

Value

External pointer to loss tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_hube ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum() ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(),

ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_loss_i *Get one head's loss scalar, before weighting*

Description

Returns the head's own loss, before it is multiplied by its `loss_weights` entry and summed into the total. `ggml_opt_loss()` returns the weighted total that is actually optimized.

Usage

```
ggml_opt_loss_i(opt_ctx, ihead)
```

Arguments

`opt_ctx`	External pointer to optimizer context
`ihead`	Output head index, 1-based

Value

External pointer to the head's loss tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_hub
ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum()
ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_loss_type_binary_cross_entropy`
Loss type: Binary Cross-Entropy

Description

Returns the constant for element-wise binary cross-entropy, `mean(-[y*log(p) + (1-y)*log(1-p)])`.

Usage

```
ggml_opt_loss_type_binary_cross_entropy()
```

Details

Unlike `ggml_opt_loss_type_cross_entropy`, which softmaxes its own input across a class axis and therefore expects logits, this one treats every output as an independent Bernoulli and expects *probabilities* – so the model's last layer should end in a sigmoid. It is the loss for multi-label targets and for a single-unit binary output. Inputs are clamped away from 0 and 1 internally, since `log(0)` would otherwise poison the whole batch.

Value

Integer constant for binary cross-entropy loss

See Also

Other optimization: `ggml_fit_opt()`, `ggml_fit_opt_gen()`, `ggml_fit_opt_multi()`, `ggml_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `ggml_opt_dataset_get_batch_head()`, `ggml_opt_dataset_init()`, `ggml_opt_dataset_labels()`, `ggml_opt_dataset_ndata()`, `ggml_opt_dataset_shuffle()`, `ggml_opt_dataset_weigh`
`ggml_opt_default_params()`, `ggml_opt_epoch()`, `ggml_opt_eval()`, `ggml_opt_fit()`, `ggml_opt_fit_multi()`, `ggml_opt_free()`, `ggml_opt_get_lr()`, `ggml_opt_grad_acc()`, `ggml_opt_init()`, `ggml_opt_init_for_fit()`, `ggml_opt_init_for_fit_multi()`, `ggml_opt_inputs()`, `ggml_opt_labels()`, `ggml_opt_loss()`, `ggml_opt_loss_i()`, `ggml_opt_loss_type_cross_entropy()`, `ggml_opt_loss_type_huber()`, `ggml_opt_loss_type_mae()`, `ggml_opt_loss_type_mean()`, `ggml_opt_loss_type_mse()`, `ggml_opt_loss_type_sum()`
`ggml_opt_loss_type_weighted_mse()`, `ggml_opt_n_loss()`, `ggml_opt_ncorrect()`, `ggml_opt_optimizer_name()`, `ggml_opt_optimizer_type_adamw()`, `ggml_opt_optimizer_type_sgd()`, `ggml_opt_outputs()`, `ggml_opt_outputs_i()`, `ggml_opt_pred()`, `ggml_opt_prepare_alloc()`, `ggml_opt_reset()`, `ggml_opt_result_accuracy()`, `ggml_opt_result_free()`, `ggml_opt_result_init()`, `ggml_opt_result_loss()`, `ggml_opt_result_loss_i()`, `ggml_opt_result_n_loss()`, `ggml_opt_result_ndata()`, `ggml_opt_result_pred()`, `ggml_opt_result_reset()`, `ggml_opt_set_lr()`, `ggml_opt_static_graphs()`, `ggml_snapshot_free()`, `ggml_trainer()`

ggml_opt_loss_type_cross_entropy

Loss type: Cross Entropy

Description

Returns the constant for cross entropy loss type. Use for classification tasks.

Usage

```
ggml_opt_loss_type_cross_entropy()
```

Value

Integer constant for cross entropy loss

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum() ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_loss_type_huber

Loss type: Huber (smooth L1)

Description

Returns the constant for the Huber loss with `delta = 1`: `0.5 * e^2` where `|e| <= 1` and `|e| - 0.5` beyond it. Quadratic near zero, so unlike MAE the gradient vanishes at the optimum, and linear far out, so unlike MSE a single outlier cannot dominate the batch.

Usage

```
ggml_opt_loss_type_huber()
```

Details

`delta` is fixed at 1, the usual default.

Value

Integer constant for Huber loss

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum() ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_loss_type_mae

Loss type: Mean Absolute Error

Description

Returns the constant for MAE (L1) loss, `mean(|pred - y|)`. Its gradient is `sgn(pred - y)`, so a far-off datapoint pulls no harder than a near one – which is what makes it more robust to outliers than MSE, at the cost of a gradient that does not vanish as the fit improves.

Usage

```
ggml_opt_loss_type_mae()
```

Value

Integer constant for MAE loss

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sur ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_loss_type_mean` *Loss type: Mean*

Description

Returns the constant for mean loss type. Custom loss - reduces outputs to mean value.

Usage

```
ggml_opt_loss_type_mean()
```

Value

Integer constant for mean loss

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(),

ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(),
ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(),
ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(),
ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(),
ggml_trainer()

`ggml_opt_loss_type_mse` *Loss type: Mean Squared Error*

Description

Returns the constant for MSE loss type. Use for regression tasks.

Usage

```
ggml_opt_loss_type_mse()
```

Value

Integer constant for MSE loss

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(),
ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(),
ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(),
ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(),
ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(),
ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(),
ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(),
ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_sur
ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(),
ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(),
ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(),
ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(),
ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(),
ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(),
ggml_trainer()

`ggml_opt_loss_type_sum` *Loss type: Sum*

Description

Returns the constant for sum loss type. Custom loss - reduces outputs to sum value.

Usage

```
ggml_opt_loss_type_sum()
```

Value

Integer constant for sum loss

See Also

Other optimization: `ggml_fit_opt()`, `ggml_fit_opt_gen()`, `ggml_fit_opt_multi()`, `ggml_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `ggml_opt_dataset_get_batch_head()`, `ggml_opt_dataset_init()`, `ggml_opt_dataset_labels()`, `ggml_opt_dataset_ndata()`, `ggml_opt_dataset_shuffle()`, `ggml_opt_dataset_weigh`
`ggml_opt_default_params()`, `ggml_opt_epoch()`, `ggml_opt_eval()`, `ggml_opt_fit()`, `ggml_opt_fit_multi()`, `ggml_opt_free()`, `ggml_opt_get_lr()`, `ggml_opt_grad_acc()`, `ggml_opt_init()`, `ggml_opt_init_for_fit()`, `ggml_opt_init_for_fit_multi()`, `ggml_opt_inputs()`, `ggml_opt_labels()`, `ggml_opt_loss()`, `ggml_opt_loss_i()`, `ggml_opt_loss_type_binary_cross_entropy()`, `ggml_opt_loss_type_cross_entropy()`, `ggml_opt_loss_type_huber()`, `ggml_opt_loss_type_mae()`, `ggml_opt_loss_type_mean()`, `ggml_opt_loss_type_mse`
`ggml_opt_loss_type_weighted_mse()`, `ggml_opt_n_loss()`, `ggml_opt_ncorrect()`, `ggml_opt_optimizer_name()`, `ggml_opt_optimizer_type_adamw()`, `ggml_opt_optimizer_type_sgd()`, `ggml_opt_outputs()`, `ggml_opt_outputs_i()`, `ggml_opt_pred()`, `ggml_opt_prepare_alloc()`, `ggml_opt_reset()`, `ggml_opt_result_accuracy()`, `ggml_opt_result_free()`, `ggml_opt_result_init()`, `ggml_opt_result_loss()`, `ggml_opt_result_loss_i()`, `ggml_opt_result_n_loss()`, `ggml_opt_result_ndata()`, `ggml_opt_result_pred()`, `ggml_opt_result_reset()`, `ggml_opt_set_lr()`, `ggml_opt_static_graphs()`, `ggml_snapshot_free()`, `ggml_trainer()`

`ggml_opt_loss_type_weighted_mse` *Loss type: Weighted Mean Squared Error*

Description

Returns the constant for per-datapoint weighted MSE loss type. Computes `sum(w * (pred - y)^2) / nelements`, where `w` is a per-sample weight supplied via `ggml_opt_dataset_weights`.

Usage

```
ggml_opt_loss_type_weighted_mse()
```

Value

Integer constant for weighted MSE loss

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_n_loss *Number of output heads in an optimizer context*

Description

Number of output heads in an optimizer context

Usage

```
ggml_opt_n_loss(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

Integer number of output heads (1 for single-output models)

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_ncorrect *Get number of correct predictions tensor*

Description

Get number of correct predictions tensor

Usage

```
ggml_opt_ncorrect(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

External pointer to ncorrect tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(),

ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse
ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_optimizer_name
ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(),
ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(),
ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(),
ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(),
ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(),
ggml_trainer()

`ggml_opt_optimizer_name` *Get optimizer name*

Description

Get optimizer name

Usage

```
ggml_opt_optimizer_name(optimizer_type)
```

Arguments

`optimizer_type`	Integer optimizer type constant

Value

Character string with optimizer name

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(),
ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(),
ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(),
ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(),
ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(),
ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(),
ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(),
ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse
ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(),
ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(),
ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(),
ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(),
ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(),
ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(),
ggml_trainer()

ggml_opt_optimizer_type_adamw

Optimizer type: AdamW

Description

Returns the constant for AdamW optimizer. Adam with weight decay - recommended for most tasks.

Usage

```
ggml_opt_optimizer_type_adamw()
```

Value

Integer constant for AdamW optimizer

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_optimizer_type_sgd

Optimizer type: SGD

Description

Returns the constant for SGD optimizer. Stochastic gradient descent - simpler but may require tuning.

Usage

```
ggml_opt_optimizer_type_sgd()
```

Value

Integer constant for SGD optimizer

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_outputs *Get outputs tensor from optimizer context*

Description

Get outputs tensor from optimizer context

Usage

```
ggml_opt_outputs(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

External pointer to outputs tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_outputs_i *Get one output head's tensors from an optimizer context*

Description

Per-head counterparts of ggml_opt_outputs() and friends, for multi-output models. ihead is 1-based; head 1 is what the single-head accessors return.

Usage

```
ggml_opt_outputs_i(opt_ctx, ihead)

ggml_opt_labels_i(opt_ctx, ihead)

ggml_opt_loss_weights_i(opt_ctx, ihead)

ggml_opt_pred_i(opt_ctx, ihead)

ggml_opt_ncorrect_i(opt_ctx, ihead)
```

Arguments

opt_ctx	External pointer to optimizer context
ihead	Output head index, 1-based

Value

External pointer to the tensor, or NULL if this head has none

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accurac ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_pred *Get predictions tensor from optimizer context*

Description

Get predictions tensor from optimizer context

Usage

```
ggml_opt_pred(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

External pointer to predictions tensor

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse

ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_prepare_alloc` *Prepare allocation for non-static graphs*

Description

Must be called before ggml_opt_alloc when not using static graphs. Sets up the optimizer context with the computation graph and input/output tensors.

Usage

```
ggml_opt_prepare_alloc(opt_ctx, ctx_compute, graph, inputs, outputs)
```

Arguments

`opt_ctx`	External pointer to optimizer context
`ctx_compute`	Compute context for temporary tensors
`graph`	Computation graph (from ggml_build_forward_expand)
`inputs`	Input tensor
`outputs`	Output tensor

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse
ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(),

ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_reset` *Reset optimizer context*

Description

Resets gradients to zero, initializes loss, and optionally resets optimizer state.

Usage

```
ggml_opt_reset(opt_ctx, optimizer = FALSE)
```

Arguments

`opt_ctx`	External pointer to optimizer context
`optimizer`	Whether to also reset optimizer state (momentum, etc.)

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh
ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse
ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_result_accuracy *Get accuracy from result*

Description

Get accuracy from result

Usage

```
ggml_opt_result_accuracy(result)
```

Arguments

result	External pointer to result object

Value

Named numeric vector with 'accuracy' and 'uncertainty'

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_result_free *Free optimization result*

Description

Free optimization result

Usage

```
ggml_opt_result_free(result)
```

Arguments

result	External pointer to result object

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_result_init *Initialize optimization result*

Description

Creates a new result object to accumulate training statistics.

Usage

```
ggml_opt_result_init()
```

Value

External pointer to result object

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_result_loss *Get loss from result*

Description

Get loss from result

Usage

```
ggml_opt_result_loss(result)
```

Arguments

result	External pointer to result object

Value

Named numeric vector with 'loss' and 'uncertainty'

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_result_loss_i` *Get one output head's loss or accuracy from a result*

Description

Per-head counterparts of `ggml_opt_result_loss()` and `ggml_opt_result_accuracy()`, which report the weighted total the optimizer minimizes. These report a single head on its own, before weighting, for the training history. `ihead` is 1-based.

Usage

```
ggml_opt_result_loss_i(result, ihead)

ggml_opt_result_accuracy_i(result, ihead)
```

Arguments

`result`	External pointer to result object
`ihead`	Output head index, 1-based

Details

Heads whose loss is not cross-entropy have no accuracy; those return `NA`.

Value

Named numeric vector with the value and its 'uncertainty'

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_result_n_loss

Number of output heads accumulated in a result

Description

Zero before the first epoch has run, so callers must treat a freshly reset result as holding no per-head data yet.

Usage

```
ggml_opt_result_n_loss(result)
```

Arguments

result	External pointer to result object

Value

Integer number of heads

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(),

ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

`ggml_opt_result_ndata` *Get number of datapoints from result*

Description

Get number of datapoints from result

Usage

```
ggml_opt_result_ndata(result)
```

Arguments

`result`	External pointer to result object

Value

Number of datapoints processed

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_result_pred *Get predictions from result*

Description

Returns the predictions as an integer vector. The length equals the number of datapoints processed.

Usage

```
ggml_opt_result_pred(result)
```

Arguments

result	External pointer to result object

Value

Integer vector of predictions

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

ggml_opt_result_reset *Reset optimization result*

Description

Reset optimization result

Usage

```
ggml_opt_result_reset(result)
```

Arguments

result	External pointer to result object

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_set_lr *Set learning rate in optimizer context*

Description

Updates the LR used for subsequent backward passes. Can be called between epochs to implement LR scheduling.

Usage

```
ggml_opt_set_lr(lr_ud, adamw_lr = NA, sgd_lr = NA)
```

Arguments

lr_ud	LR userdata pointer (from 'ggml_opt_init_for_fit()$lr_ud')
adamw_lr	New AdamW learning rate (NA to keep current)
sgd_lr	New SGD learning rate (NA to keep current)

Value

NULL invisibly

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_static_graphs(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_static_graphs

Check if using static graphs

Description

Check if using static graphs

Usage

```
ggml_opt_static_graphs(opt_ctx)
```

Arguments

opt_ctx	External pointer to optimizer context

Value

Logical indicating if graphs are statically allocated

See Also

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_snapshot_free(), ggml_trainer()

---

ggml_opt_step_adamw *AdamW Optimizer Step (Graph Node)*

Description

Builds a node that applies one AdamW step when the graph is computed:

```
m <- beta1 * m + (1 - beta1) * grad
v <- beta2 * v + (1 - beta2) * grad^2
a <- a * (1 - alpha * wd) - alpha * (m * beta1h) / (sqrt(v * beta2h) + eps)
```

`a`, `m` and `v` are updated *in place*; the returned tensor is a view of `a`. Because one kernel reads and writes each element, the update is free of the read-before-write ordering that a chain of separate ops would need. With `wd = 0` the decay factor is exactly 1.

Usage

```
ggml_opt_step_adamw(ctx, a, grad, m, v, adamw_params)
```

Arguments

`ctx`	GGML context
`a`	Weight tensor, F32, contiguous. It need not be marked with ggml_set_param: the flag ggml asserts is set only while the node is built, then the tensor's own flags are restored (a lasting PARAM flag would turn the weight into a node of every later graph)
`grad`	Gradient, same shape as `a`, F32
`m`	First moment, same shape as `a`, F32
`v`	Second moment, same shape as `a`, F32
`adamw_params`	F32 tensor of 7 elements: `alpha`, `beta1`, `beta2`, `eps`, `wd`, `beta1h`, `beta2h`

Details

Precision: the kernel receives `beta2` as a float and forms `1 - beta2` itself, so the weight of `grad^2` in `v` is `1 - fl(beta2)`, not `1 - beta2`. Its relative error is 9.5e-7 for 0.99, 1.29e-5 for 0.999 and 1.66e-4 for 0.9999 – the closer `beta2` is to 1, the larger. The effect on the weight is at most about half of that. Expect agreement with a double-precision Adam to that level, not to the last bit.

Bias correction is passed in, not derived: for step `t`, `beta1h = 1 / (1 - beta1^t)` and `beta2h = 1 / (1 - beta2^t)`.

Only F32 is supported (the CPU and Vulkan kernels are F32-only); any other type or a shape mismatch is an R error.

Value

A view of `a`; expand it into a graph to run the step

See Also

`ggml_set_param`, `optimizer_adam`

`ggml_out_prod` *Outer Product (Graph)*

Description

Computes the outer product of two vectors: C = a * b^T For vectors a[m] and b[n], produces matrix C[m, n].

Usage

```
ggml_out_prod(ctx, a, b)
```

Arguments

`ctx`	GGML context
`a`	First vector tensor
`b`	Second vector tensor

Value

Matrix tensor representing the outer product

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 3)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3))
ggml_set_f32(b, c(1, 2, 3, 4))
c <- ggml_out_prod(ctx, a, b)  # Result: 3x4 matrix
graph <- ggml_build_forward_expand(ctx, c)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_pad *Pad Tensor with Zeros (Graph)*

Description

Pads tensor dimensions with zeros on the right side. Useful for aligning tensor sizes in attention operations.

Usage

```
ggml_pad(ctx, a, p0 = 0L, p1 = 0L, p2 = 0L, p3 = 0L)
```

Arguments

ctx	GGML context
a	Input tensor to pad
p0	Padding for dimension 0 (default 0)
p1	Padding for dimension 1 (default 0)
p2	Padding for dimension 2 (default 0)
p3	Padding for dimension 3 (default 0)

Value

Padded tensor with shape [ne0+p0, ne1+p1, ne2+p2, ne3+p3]

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 5, 3)
ggml_set_f32(a, 1:15)
# Pad to 8x4
b <- ggml_pad(ctx, a, 3, 1)  # Add 3 zeros to dim0, 1 to dim1
graph <- ggml_build_forward_expand(ctx, b)
ggml_graph_compute(ctx, graph)
# Result shape: [8, 4]
ggml_free(ctx)
```

ggml_pad_ext *Pad a tensor with zeros on both sides*

Description

Like ggml_pad, but each dimension can also be padded before its first element.

Usage

```
ggml_pad_ext(ctx, a, lp = c(0L, 0L, 0L, 0L), rp = c(0L, 0L, 0L, 0L))
```

Arguments

ctx	GGML context
a	Input tensor to pad
lp	Integer vector of length 4: zeros added before dimensions 0..3
rp	Integer vector of length 4: zeros added after dimensions 0..3

Value

Padded tensor with shape [ne0+lp0+rp0, ne1+lp1+rp1, ...]

ggml_pad_reflect_1d *Reflective 1D Padding (Graph)*

Description

Pads the first dimension of a tensor using reflection of its values.

Usage

```
ggml_pad_reflect_1d(ctx, a, p0, p1)
```

Arguments

ctx	GGML context
a	Input tensor
p0	Left padding
p1	Right padding

Value

Padded tensor

ggml_permute *Permute Tensor Dimensions (Graph)*

Description

Permutes the tensor dimensions according to specified axes. CRITICAL for attention mechanisms in transformers.

Usage

```
ggml_permute(ctx, a, axis0, axis1, axis2, axis3)
```

Arguments

ctx	GGML context
a	Input tensor
axis0	New position for axis 0
axis1	New position for axis 1
axis2	New position for axis 2
axis3	New position for axis 3

Value

Permuted tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Create 4D tensor: (2, 3, 4, 5)
t <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, 2, 3, 4, 5)
# Swap axes 0 and 1: result shape (3, 2, 4, 5)
t_perm <- ggml_permute(ctx, t, 1, 0, 2, 3)
ggml_free(ctx)
```

ggml_pool_1d *1D Pooling (Graph)*

Description

Applies 1D pooling operation for downsampling.

Usage

```
ggml_pool_1d(ctx, a, op, k0, s0 = k0, p0 = 0L)

GGML_OP_POOL_MAX

GGML_OP_POOL_AVG
```

Arguments

ctx	GGML context
a	Input tensor
op	Pool operation constant (see details)
k0	Kernel size (window size)
s0	Stride (default = k0 for non-overlapping windows)
p0	Padding (default 0)

Format

An object of class integer of length 1.

An object of class integer of length 1.

Details

Pool operation constants:

- GGML_OP_POOL_MAX (0): Max pooling - takes maximum value in each window
- GGML_OP_POOL_AVG (1): Average pooling - takes mean of values in each window

Value

Pooled tensor with reduced dimensions

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 8)
ggml_set_f32(a, c(1, 3, 2, 4, 5, 2, 8, 1))

# Max pooling with kernel 2, stride 2
max_pool <- ggml_pool_1d(ctx, a, GGML_OP_POOL_MAX, k0 = 2)
# Result: [3, 4, 5, 8] (max of each pair)

# Average pooling with kernel 2, stride 2
avg_pool <- ggml_pool_1d(ctx, a, GGML_OP_POOL_AVG, k0 = 2)
# Result: [2, 3, 3.5, 4.5] (mean of each pair)

ggml_free(ctx)
```

ggml_pool_2d *2D Pooling (Graph)*

Description

Applies 2D pooling operation.

Usage

```
ggml_pool_2d(ctx, a, op, k0, k1, s0 = k0, s1 = k1, p0 = 0, p1 = 0)
```

Arguments

ctx	GGML context
a	Input tensor
op	Pool operation: GGML_OP_POOL_MAX (0) or GGML_OP_POOL_AVG (1)
k0	Kernel size dimension 0
k1	Kernel size dimension 1
s0	Stride dimension 0 (default = k0)
s1	Stride dimension 1 (default = k1)
p0	Padding dimension 0 (default 0)
p1	Padding dimension 1 (default 0)

Value

Pooled tensor

ggml_pop_layer *Remove the Last Layer from a Sequential Model*

Description

Removes the last layer from the model. The model must not be compiled.

Usage

```
ggml_pop_layer(model)
```

Arguments

model	A ggml_sequential_model object

Value

The model with the last layer removed.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(64, activation = "relu") |>
  ggml_layer_dense(10, activation = "softmax")

model <- ggml_pop_layer(model)
length(model$layers)  # 1
```

---

ggml_pp_dp_forward *Pipeline + data parallel forward pass (PP x DP)*

Description

Runs data parallelism over the batch across replicas of a pipeline-parallel model: the batch is split into contiguous shards, one per replica, and each replica runs the whole model as a device-by-layer **pipeline** on its own set of GPUs (via ggml_pp_forward). This is the pipeline counterpart of ggml_tp_dp_forward — use it when the model is too large for one card (split it by layers) and you also want to raise throughput by replicating the pipeline across independent GPU sets.

Usage

```
ggml_pp_dp_forward(
  make_stages,
  x,
  replicas,
  out_ncol,
  mem_per_stage = 16L * 1024L * 1024L
)
```

Arguments

make_stages	A function function(devices, m_shard) returning the list of pipeline stage descriptors (see ggml_pp_forward) for a replica whose pipeline runs on devices (an integer GPU-index vector) and whose batch shard has m_shard samples. The stages' in_shape / weight setup must be built for that m_shard.
x	Input activation matrix, M x K (M samples, K features). Its rows (the batch) are split across the replicas.
replicas	A list of integer vectors, each the GPU-index pipeline for one replica, e.g. list(c(0, 1), c(2, 3)) for PP=2 x DP=2.
out_ncol	Number of columns N of the per-sample output (the final stage produces an N x m_shard result per replica; the gathered result is M x N).
mem_per_stage	Bytes of ggml context metadata per stage (see ggml_pp_forward).

Details

On a 4-GPU box the natural layout is **PP=2 x DP=2**: replica A pipelines its layers across GPUs `c(0, 1)`, replica B across `c(2, 3)`, and the two replicas each handle half the batch with no cross-replica traffic.

Value

The `M` x `N` result matrix, the batch-concatenation of each replica's pipeline output.

See Also

`ggml_pp_forward`, `ggml_tp_dp_forward`.

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 4) {
  K <- 64L
  W1 <- matrix(rnorm(K * K), K); W2 <- matrix(rnorm(K * K), K)
  make_stages <- function(devices, m) {
    mk <- function(dev, Wt, relu) list(
      device = dev, in_shape = c(K, m),
      build = function(ctx, input) {
        w <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, K, K)
        z <- ggml_mul_mat(ctx, w, input)
        list(output = if (relu) ggml_relu(ctx, z) else z,
             set_weights = function() ggml_backend_tensor_set_data(w, as.numeric(Wt)))
      })
    list(mk(devices[1], W1, TRUE), mk(devices[2], W2, FALSE))
  }
  X <- matrix(rnorm(8 * K), nrow = 8)   # batch of 8
  Y <- ggml_pp_dp_forward(make_stages, X, replicas = list(c(0, 1), c(2, 3)),
                          out_ncol = K)
}
```

---

ggml_pp_forward *Pipeline-parallel forward pass across per-device layer stages*

Description

Runs a forward pass split BY LAYERS across devices (pipeline parallelism), the complement of `ggml_vulkan_split_mul_mat`'s split-by-matrix tensor parallelism. Each `stage` owns a contiguous block of the model's layers on one GPU; the activation tensor is handed from one stage to the next exactly once per pass (a single cross-device copy), versus TP's per-layer gather. This suits models too large for one card's VRAM: at ~1 GB/s host staging the single handoff costs only ~10-20 ms per pass.

Usage

```
ggml_pp_forward(stages, x, out_shape, mem_per_stage = 16L * 1024L * 1024L)
```

Arguments

stages	A list of stage descriptors (see Details), in pipeline order.
x	A numeric vector/array of input activations for the first stage, laid out in ggml (column-major) order matching `stages[[1]]$in_shape`.
out_shape	Integer vector: the ggml ne shape of the final stage's output, used to size the returned vector.
mem_per_stage	Bytes of ggml context metadata to reserve per stage (default 16 MiB) — raise it for stages with very many ops.

Details

Each stage is a list with:

`device` Physical GPU index (0-based) the stage runs on.

`build` A function `function(ctx, input)` that builds this stage's sub-graph from the `input` tensor using the `ggml_*` ops and returns either the output tensor, or a list `list(output = <tensor>, set_weights = function() ...)`. The `set_weights` closure (if given) is called AFTER the stage's tensors are allocated on the device, and is where weight tensors created inside `build` get their values (via `ggml_backend_tensor_set_data`).

`in_shape` Integer vector: the ggml ne shape of this stage's input tensor (fastest dim first). For stage 1 this must match the supplied `x`; for later stages it must match the previous stage's output shape.

Value

A numeric vector of the final stage's output activations (ggml order).

See Also

`ggml_tp_dp_forward` for the tensor-parallel counterpart.

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 2) {
  K <- 32L; M <- 4L
  W1 <- matrix(rnorm(K * K), K); W2 <- matrix(rnorm(K * K), K)
  stage <- function(dev, Wt) list(
    device = dev, in_shape = c(K, M),
    build = function(ctx, input) {
      w <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, K, K)
      ggml_mul_mat(ctx, w, input)   # (weights set inside build via attr, see vignette)
    })
  # See inst/examples/pp_pipeline.R for a complete runnable stage definition.
}
```

ggml_predict.ggml_functional_model
Get Predictions from a Trained Model

Description

Runs forward pass on input data and returns prediction probabilities (or raw output values for regression). Unlike `ggml_evaluate()`, this does not require labels.

Usage

```
## S3 method for class 'ggml_functional_model'
ggml_predict(model, x, batch_size = 32L, matmul_precision = NULL, ...)

ggml_predict(model, ...)

## S3 method for class 'ggml_sequential_model'
ggml_predict(model, x, batch_size = 32L, matmul_precision = NULL, ...)
```

Arguments

`model`	A trained ggml_sequential_model
`x`	Input data (matrix or array)
`batch_size`	Batch size for inference
`matmul_precision`	`"default"` or `"f32"` for this call; `NULL` (the default) takes the model's `predict_precision` from `ggml_compile`. See the section on precision.
`...`	Additional arguments (ignored).

Value

Matrix of predictions with shape `[N, output_units]`

Precision

On a Vulkan device with fp16 support the matrix multiplies keep their tiles in f16 by default. Results then carry a relative error near 1e-3 that depends on the batch size (a single row takes an exact f32 path, larger batches the tiled one), so the same input can predict slightly differently in a batch of 1 and a batch of 64. `"f32"` marks every multiply `GGML_PREC_F32` and selects f32 tiles: the result then matches the CPU to about 1e-6, at a cost in matmul throughput that depends on the device and driver. The CPU backend is exact either way.

Which setting wins: the `matmul_precision` argument of the call, then the model's `predict_precision`, then `"default"`.

ggml_predict_classes *Predict Classes from a Trained Model*

Description

Returns predicted class indices (1-based) by applying argmax to the output of `ggml_predict()`.

Usage

```
ggml_predict_classes(model, x, batch_size = 32L)
```

Arguments

`model`	A trained ggml_sequential_model
`x`	Input data (matrix or array)
`batch_size`	Batch size for inference

Value

Integer vector of predicted class indices (1-based)

ggml_print_mem_status *Print Context Memory Status*

Description

Helper to print memory usage information

Usage

```
ggml_print_mem_status(ctx)
```

Arguments

`ctx`	GGML context

Value

List with total, used, free memory (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_print_mem_status(ctx)
ggml_free(ctx)
```

ggml_print_objects *Print Objects in Context*

Description

Debug function to print all objects (tensors) in the context

Usage

```
ggml_print_objects(ctx)
```

Arguments

ctx	GGML context

Value

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_print_objects(ctx)
ggml_free(ctx)
```

ggml_quant_block_info *Get Quantization Block Info*

Description

Returns information about a quantization type including name, type size, block size, and whether it's quantized.

Usage

```
ggml_quant_block_info(type)
```

Arguments

type	GGML type constant

Value

List with type_name, type_size, block_size, is_quantized

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

ggml_quantize_chunk *Quantize Data Chunk*

Description

Quantizes a chunk of floating-point data to a lower precision format.

Usage

```
ggml_quantize_chunk(type, src, nrows, n_per_row)
```

Arguments

type	Target GGML type (e.g., GGML_TYPE_Q4_0)
src	Source numeric vector (F32 data)
nrows	Number of rows
n_per_row	Number of elements per row

Value

Raw vector containing quantized data

Examples

```
# Quantize 256 floats to Q8_0 (block size 32)
data <- rnorm(256)
quantized <- ggml_quantize_chunk(GGML_TYPE_Q8_0, data, 1, 256)
ggml_quantize_free()  # Clean up
```

ggml_quantize_free *Free Quantization Resources*

Description

Frees any memory allocated by quantization. Call at end of program to avoid memory leaks.

Usage

```
ggml_quantize_free()
```

Value

NULL invisibly

ggml_quantize_init *Initialize Quantization Tables*

Description

Initializes quantization tables for a given type. Called automatically by ggml_quantize_chunk, but can be called manually.

Usage

```
ggml_quantize_init(type)
```

Arguments

type	GGML type (e.g., GGML_TYPE_Q4_0)

Value

NULL invisibly

ggml_quantize_requires_imatrix *Check if Quantization Requires Importance Matrix*

Description

Some quantization types require an importance matrix for optimal quality.

Usage

```
ggml_quantize_requires_imatrix(type)
```

Arguments

type	GGML type

Value

TRUE if importance matrix is required

ggml_reglu *ReGLU (ReLU Gated Linear Unit) (Graph)*

Description

Creates a graph node for ReGLU operation. ReGLU uses ReLU as the activation function on the first half.

Usage

```
ggml_reglu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (first dimension must be even)

Details

Formula: output = ReLU(x) * gate

Value

Tensor with half the first dimension of input

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 8, 3)
ggml_set_f32(a, rnorm(24))
r <- ggml_reglu(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # Shape: 4x3
ggml_free(ctx)
```

---

ggml_reglu_split *ReGLU Split (Graph)*

Description

Creates a graph node for ReGLU with separate input and gate tensors.

Usage

```
ggml_reglu_split(ctx, a, b)
```

Arguments

ctx	GGML context
a	Input tensor (the values to be gated)
b	Gate tensor (same shape as a)

Details

Formula: output = ReLU(a) * b

Value

Tensor with same shape as input tensors

ggml_relu *ReLU Activation (Graph)*

Description

Creates a graph node for ReLU (Rectified Linear Unit) activation: max(0, x)

Usage

```
ggml_relu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the ReLU operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
result <- ggml_relu(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

ggml_relu_inplace *ReLU Activation In-place (Graph)*

Description

Creates a graph node for in-place ReLU activation: max(0, x)

Usage

```
ggml_relu_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with ReLU applied

`ggml_repeat` *Repeat (Graph)*

Description

Creates a graph node that repeats tensor 'a' to match shape of tensor 'b'.

Usage

```
ggml_repeat(ctx, a, b)
```

Arguments

`ctx`	GGML context
`a`	Tensor to repeat
`b`	Target tensor (defines output shape)

Value

Tensor with repeated values

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 1, 2)
ggml_set_f32(a, c(1, 2))
b <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 2)
result <- ggml_repeat(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [1, 1, 1, 2, 2, 2]
ggml_free(ctx)
```

ggml_repeat_back *Repeat Backward (Graph)*

Description

Backward pass for repeat operation - sums repetitions back to original shape. Used for gradient computation during training.

Usage

```
ggml_repeat_back(ctx, a, b)
```

Arguments

ctx	GGML context
a	Input tensor (gradients from repeated tensor)
b	Target shape tensor (original tensor before repeat)

Value

Tensor with summed gradients matching shape of b

ggml_reset *Reset GGML Context*

Description

Clears all tensor allocations in the context memory pool. The context can be reused without recreating it. This is more efficient than free + init for temporary operations.

Usage

```
ggml_reset(ctx)
```

Arguments

ctx	GGML context pointer

Value

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
ggml_reset(ctx)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 200)
ggml_free(ctx)
```

---

ggml_reshape_1d *Reshape to 1D (Graph)*

Description

Reshapes tensor to 1D with ne0 elements

Usage

```
ggml_reshape_1d(ctx, a, ne0)
```

Arguments

ctx	GGML context
a	Input tensor
ne0	Size of dimension 0

Value

Reshaped tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 4)
ggml_set_f32(a, 1:12)
result <- ggml_reshape_1d(ctx, a, 12)
ggml_free(ctx)
```

ggml_reshape_2d *Reshape to 2D (Graph)*

Description

Reshapes tensor to 2D with shape (ne0, ne1)

Usage

```
ggml_reshape_2d(ctx, a, ne0, ne1)
```

Arguments

ctx	GGML context
a	Input tensor
ne0	Size of dimension 0
ne1	Size of dimension 1

Value

Reshaped tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 12)
ggml_set_f32(a, 1:12)
result <- ggml_reshape_2d(ctx, a, 3, 4)
ggml_free(ctx)
```

ggml_reshape_3d *Reshape to 3D (Graph)*

Description

Reshapes tensor to 3D with shape (ne0, ne1, ne2)

Usage

```
ggml_reshape_3d(ctx, a, ne0, ne1, ne2)
```

Arguments

ctx	GGML context
a	Input tensor
ne0	Size of dimension 0
ne1	Size of dimension 1
ne2	Size of dimension 2

Value

Reshaped tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 24)
ggml_set_f32(a, 1:24)
result <- ggml_reshape_3d(ctx, a, 2, 3, 4)
ggml_free(ctx)
```

---

ggml_reshape_4d *Reshape to 4D (Graph)*

Description

Reshapes tensor to 4D with shape (ne0, ne1, ne2, ne3)

Usage

```
ggml_reshape_4d(ctx, a, ne0, ne1, ne2, ne3)
```

Arguments

ctx	GGML context
a	Input tensor
ne0	Size of dimension 0
ne1	Size of dimension 1
ne2	Size of dimension 2
ne3	Size of dimension 3

Value

Reshaped tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 120)
ggml_set_f32(a, 1:120)
result <- ggml_reshape_4d(ctx, a, 2, 3, 4, 5)
ggml_free(ctx)
```

---

ggml_result *Construct a single-cell result*

Description

A `ggml_result` is the contract object returned by the dispatch layer and consumed by the injection layer. The embedding is stored cell-by-component (cells in rows), ready to drop into `reducedDim()` / a Seurat reduction.

Usage

```
ggml_result(embedding, metadata = list(), timings = numeric(0))
```

Arguments

`embedding`	A numeric matrix, cells in rows, components in columns.
`metadata`	Named list (e.g. `stdev`, `loadings`, backend used).
`timings`	Named numeric vector of elapsed seconds per stage.

Value

An object of class `ggml_result`.

---

ggml_rms_norm *RMS Normalization (Graph)*

Description

Creates a graph node for RMS (Root Mean Square) normalization. Normalizes by x / sqrt(mean(x^2) + eps). CRITICAL for LLaMA models.

Usage

```
ggml_rms_norm(ctx, a, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Input tensor
eps	Epsilon value for numerical stability (default: 1e-5)

Value

Tensor representing the RMS normalization operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_rms_norm(ctx, a, eps = 1e-5)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)
# sqrt(mean(output^2)) should be ~1
ggml_free(ctx)
```

---

ggml_rms_norm_back *RMS Norm Backward (Graph)*

Description

Creates a graph node for backward pass of RMS normalization. Used in training for computing gradients.

Usage

```
ggml_rms_norm_back(ctx, a, b, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Gradient tensor (dy from the forward output)
b	Input tensor (x from the forward pass)
eps	Epsilon for numerical stability (default 1e-5)

Value

Tensor representing the gradient with respect to input

ggml_rms_norm_inplace *RMS Normalization In-place (Graph)*

Description

Creates a graph node for in-place RMS normalization. Returns a view of the input tensor. CRITICAL for LLaMA models when memory efficiency is important.

Usage

```
ggml_rms_norm_inplace(ctx, a, eps = 1e-05)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)
eps	Epsilon value for numerical stability (default: 1e-5)

Value

View of input tensor with RMS normalization applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_rms_norm_inplace(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_roll *Roll (Graph)*

Description

Circularly shifts tensor elements along dimensions 0..3.

Usage

```
ggml_roll(ctx, a, shift0 = 0L, shift1 = 0L, shift2 = 0L, shift3 = 0L)
```

Arguments

ctx	GGML context
a	Input tensor
shift0, shift1, shift2, shift3	Shift amount along each dimension

Value

Rolled tensor

---

ggml_rope *Rotary Position Embedding (Graph)*

Description

Creates a graph node for RoPE (Rotary Position Embedding). RoPE is the dominant position encoding method in modern LLMs like LLaMA, Mistral, and many others.

Usage

```
ggml_rope(ctx, a, b, n_dims, mode = 0L)
```

Arguments

ctx	GGML context
a	Input tensor of shape [head_dim, n_head, seq_len, batch]
b	Position tensor (int32) of shape [seq_len] containing position indices
n_dims	Number of dimensions to apply rotation to (usually head_dim)
mode	RoPE mode: GGML_ROPE_TYPE_NORM (0), GGML_ROPE_TYPE_NEOX (2), etc.

Details

RoPE encodes position information by rotating pairs of dimensions in the embedding space. The rotation angle depends on position and dimension index.

Key benefits of RoPE: - Relative position information emerges naturally from rotation - Better extrapolation to longer sequences than absolute embeddings - No additional parameters needed

Value

Tensor with same shape as input, with rotary embeddings applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Query tensor: head_dim=8, n_head=4, seq_len=16, batch=1
q <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, 8, 4, 16, 1)
ggml_set_f32(q, rnorm(8 * 4 * 16))
# Position indices
pos <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 16)
ggml_set_i32(pos, 0:15)
# Apply RoPE
q_rope <- ggml_rope(ctx, q, pos, 8, GGML_ROPE_TYPE_NORM)
graph <- ggml_build_forward_expand(ctx, q_rope)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_rope_ext *Extended RoPE with Frequency Scaling (Graph)*

Description

Creates a graph node for extended RoPE with frequency scaling parameters. Supports context extension techniques like YaRN, Linear Scaling, etc.

Usage

```
ggml_rope_ext(
  ctx,
  a,
  b,
  c = NULL,
  n_dims,
  mode = 0L,
  n_ctx_orig = 0L,
  freq_base = 10000,
  freq_scale = 1,
  ext_factor = 0,
  attn_factor = 1,
  beta_fast = 32,
  beta_slow = 1
)
```

Arguments

ctx	GGML context
a	Input tensor
b	Position tensor (int32)
c	Optional frequency factors tensor (NULL for default)

n_dims	Number of dimensions to apply rotation to
mode	RoPE mode
n_ctx_orig	Original context length the model was trained on
freq_base	Base frequency for RoPE (default 10000 for most models)
freq_scale	Frequency scale factor (1.0 = no scaling)
ext_factor	YaRN extension factor (0.0 to disable)
attn_factor	Attention scale factor (typically 1.0)
beta_fast	YaRN parameter for fast dimensions
beta_slow	YaRN parameter for slow dimensions

Details

This extended version supports various context extension techniques:

- **Linear Scaling**: Set freq_scale = original_ctx / new_ctx - **YaRN**: Set ext_factor > 0 with appropriate beta_fast/beta_slow - **NTK-aware**: Adjust freq_base for NTK-style scaling

Common freq_base values: - LLaMA 1/2: 10000 - LLaMA 3: 500000 - Mistral: 10000 - Phi-3: 10000

Value

Tensor with extended RoPE applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
q <- ggml_new_tensor_4d(ctx, GGML_TYPE_F32, 64, 8, 32, 1)
ggml_set_f32(q, rnorm(64 * 8 * 32))
pos <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 32)
ggml_set_i32(pos, 0:31)
# Standard RoPE with default freq_base
q_rope <- ggml_rope_ext(ctx, q, pos, NULL,
                        n_dims = 64, mode = 0L,
                        n_ctx_orig = 4096,
                        freq_base = 10000, freq_scale = 1.0,
                        ext_factor = 0.0, attn_factor = 1.0,
                        beta_fast = 32, beta_slow = 1)
graph <- ggml_build_forward_expand(ctx, q_rope)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_rope_ext_back *RoPE Extended Backward (Graph)*

Description

Backward pass for extended RoPE (Rotary Position Embedding). Used during training to compute gradients through RoPE.

Usage

```
ggml_rope_ext_back(
  ctx,
  a,
  b,
  c = NULL,
  n_dims,
  mode = 0L,
  n_ctx_orig = 0L,
  freq_base = 10000,
  freq_scale = 1,
  ext_factor = 0,
  attn_factor = 1,
  beta_fast = 32,
  beta_slow = 1
)
```

Arguments

ctx	GGML context
a	Gradient tensor from upstream (gradients of ggml_rope_ext result)
b	Position tensor (same as forward pass)
c	Optional frequency factors tensor (NULL for default)
n_dims	Number of dimensions for rotation
mode	RoPE mode
n_ctx_orig	Original context length
freq_base	Base frequency
freq_scale	Frequency scale factor
ext_factor	Extension factor (YaRN)
attn_factor	Attention factor
beta_fast	YaRN fast beta
beta_slow	YaRN slow beta

Value

Gradient tensor for the input

`ggml_rope_ext_inplace` *Extended RoPE Inplace (Graph)*

Description

Creates a graph node for extended RoPE, modifying input tensor in place. Returns a view of the input tensor.

Usage

```
ggml_rope_ext_inplace(
  ctx,
  a,
  b,
  c = NULL,
  n_dims,
  mode = 0L,
  n_ctx_orig = 0L,
  freq_base = 10000,
  freq_scale = 1,
  ext_factor = 0,
  attn_factor = 1,
  beta_fast = 32,
  beta_slow = 1
)
```

Arguments

`ctx`	GGML context
`a`	Input tensor
`b`	Position tensor (int32)
`c`	Optional frequency factors tensor (NULL for default)
`n_dims`	Number of dimensions to apply rotation to
`mode`	RoPE mode
`n_ctx_orig`	Original context length the model was trained on
`freq_base`	Base frequency for RoPE (default 10000 for most models)
`freq_scale`	Frequency scale factor (1.0 = no scaling)
`ext_factor`	YaRN extension factor (0.0 to disable)
`attn_factor`	Attention scale factor (typically 1.0)
`beta_fast`	YaRN parameter for fast dimensions
`beta_slow`	YaRN parameter for slow dimensions

Value

View of input tensor with RoPE applied in place

See Also

Other rope: `ggml_rope_multi()`, `ggml_rope_multi_inplace()`

`ggml_rope_inplace` *Rotary Position Embedding In-place (Graph)*

Description

In-place version of ggml_rope. Returns a view of the input tensor.

Usage

```
ggml_rope_inplace(ctx, a, b, n_dims, mode = 0L)
```

Arguments

`ctx`	GGML context
`a`	Input tensor (will be modified in-place)
`b`	Position tensor (int32)
`n_dims`	Number of dimensions to apply rotation to
`mode`	RoPE mode

Value

View of input tensor with RoPE applied

`ggml_rope_multi` *Multi-RoPE for Vision Models (Graph)*

Description

Creates a graph node for multi-dimensional RoPE (MRoPE) used in vision transformers. Supports separate rotation for different positional dimensions (e.g., height, width, time).

Usage

```
ggml_rope_multi(
  ctx,
  a,
  b,
  c = NULL,
  n_dims,
  sections = c(0L, 0L, 0L, 0L),
  mode = 0L,
  n_ctx_orig = 0L,
  freq_base = 10000,
  freq_scale = 1,
  ext_factor = 0,
  attn_factor = 1,
  beta_fast = 32,
  beta_slow = 1
)
```

Arguments

ctx	GGML context
a	Input tensor
b	Position tensor (int32)
c	Optional frequency factors tensor (NULL for default)
n_dims	Number of dimensions to apply rotation to
sections	Integer vector of length 4 specifying dimension sections for MRoPE
mode	RoPE mode
n_ctx_orig	Original context length the model was trained on
freq_base	Base frequency for RoPE (default 10000 for most models)
freq_scale	Frequency scale factor (1.0 = no scaling)
ext_factor	YaRN extension factor (0.0 to disable)
attn_factor	Attention scale factor (typically 1.0)
beta_fast	YaRN parameter for fast dimensions
beta_slow	YaRN parameter for slow dimensions

Value

Tensor with multi-dimensional RoPE applied

See Also

Other rope: ggml_rope_ext_inplace(), ggml_rope_multi_inplace()

ggml_rope_multi_back *Multi-RoPE Backward (Graph)*

Description

Gradient node for ggml_rope_multi. Takes the same geometry arguments as the forward op, which is what makes the rotation invertible.

Usage

```
ggml_rope_multi_back(
  ctx,
  a,
  b,
  c = NULL,
  n_dims,
  sections = c(0L, 0L, 0L, 0L),
  mode = 0L,
  n_ctx_orig = 0L,
  freq_base = 10000,
  freq_scale = 1,
  ext_factor = 0,
  attn_factor = 1,
  beta_fast = 32,
  beta_slow = 1
)
```

Arguments

ctx	GGML context
a	Input tensor
b	Position tensor (int32)
c	Optional frequency factors tensor (NULL for default)
n_dims	Number of dimensions to apply rotation to
sections	Integer vector of length 4 specifying dimension sections for MRoPE
mode	RoPE mode
n_ctx_orig	Original context length the model was trained on
freq_base	Base frequency for RoPE (default 10000 for most models)
freq_scale	Frequency scale factor (1.0 = no scaling)
ext_factor	YaRN extension factor (0.0 to disable)
attn_factor	Attention scale factor (typically 1.0)
beta_fast	YaRN parameter for fast dimensions
beta_slow	YaRN parameter for slow dimensions

Value

Gradient w.r.t. the rotated input

See Also

ggml_rope_multi

`ggml_rope_multi_inplace` *Multi-RoPE Inplace (Graph)*

Description

Creates a graph node for multi-dimensional RoPE, modifying input in place.

Usage

```
ggml_rope_multi_inplace(
  ctx,
  a,
  b,
  c = NULL,
  n_dims,
  sections = c(0L, 0L, 0L, 0L),
  mode = 0L,
  n_ctx_orig = 0L,
  freq_base = 10000,
  freq_scale = 1,
  ext_factor = 0,
  attn_factor = 1,
  beta_fast = 32,
  beta_slow = 1
)
```

Arguments

ctx	GGML context
a	Input tensor
b	Position tensor (int32)
c	Optional frequency factors tensor (NULL for default)
n_dims	Number of dimensions to apply rotation to
sections	Integer vector of length 4 specifying dimension sections for MRoPE
mode	RoPE mode
n_ctx_orig	Original context length the model was trained on

freq_base	Base frequency for RoPE (default 10000 for most models)
freq_scale	Frequency scale factor (1.0 = no scaling)
ext_factor	YaRN extension factor (0.0 to disable)
attn_factor	Attention scale factor (typically 1.0)
beta_fast	YaRN parameter for fast dimensions
beta_slow	YaRN parameter for slow dimensions

Value

View of input tensor with MRoPE applied in place

See Also

Other rope: ggml_rope_ext_inplace(), ggml_rope_multi()

---

ggml_round *Round (Graph)*

Description

Creates a graph node for element-wise rounding: round(x)

Usage

```
ggml_round(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the round operation

ggml_round_inplace *Round In-place (Graph)*

Description

Creates a graph node for in-place element-wise rounding.

Usage

```
ggml_round_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with rounded values

ggml_run *Run a single-cell task on the GGML backend*

Description

Validates a ggml_task against ggml_ops_registry, resolves the compute device, invokes the registered engine and returns a ggml_result. This is the dispatch layer: the only place that decides GPU vs CPU. device = "auto" (the task default) uses Vulkan when a GPU is available and CPU otherwise; device = "vulkan" on a machine with no GPU degrades to CPU with a message rather than failing.

Usage

```
ggml_run(task, backend = NULL, ...)
```

Arguments

task	A ggml_task.
backend	Optional override of task$device: "auto" (default), "vulkan" or "cpu".
...	Additional parameters merged over task$params and passed to the engine.

Value

A ggml_result.

See Also

ggml_task, ggml_ops_registry

---

ggml_rwkv_output *Views Into a Packed RWKV / GLA Result*

Description

ggml_rwkv_wkv6, ggml_rwkv_wkv7 and ggml_gated_linear_attn all return one [S * H, n_tokens + S * n_seqs] tensor holding the per-token outputs followed by the final state. These helpers view either half, so the column arithmetic lives in one place rather than at every call site.

Usage

```
ggml_rwkv_output(ctx, result, k)

ggml_rwkv_state(ctx, result, k, state)
```

Arguments

ctx	GGML context
result	The tensor returned by one of the RWKV / GLA ops
k	The k passed to that op (gives S, H and the token count)
state	The state passed to that op (gives the sequence count)

Details

Both are graph views sharing memory with result and add no computation. k must be the same tensor the op was built with, since S, H and n_tokens are read from it.

Value

For ggml_rwkv_output(), a tensor [S * H, n_tokens]

For ggml_rwkv_state(), a tensor [S * H, S * n_seqs]

See Also

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

ggml_rwkv_wkv6 *RWKV-6 WKV Recurrence*

Description

The weighted key-value recurrence of RWKV-6: a linear-attention-style scan whose decay is per-token (`td`) on top of a per-channel bonus (`tf`). Runs in O(n_tokens) rather than the O(n_tokens^2) of ordinary attention, which is what makes RWKV a recurrent model at inference time.

Usage

```
ggml_rwkv_wkv6(ctx, k, v, r, tf, td, state)
```

Arguments

`ctx`	GGML context
`k`	Keys, `[S, H, n_tokens]`
`v`	Values, `[S, H, n_tokens]`
`r`	Receptance, `[S, H, n_tokens]`
`tf`	Per-channel decay bonus, `[S, H]`
`td`	Per-token decay, `[S, H, n_tokens]`
`state`	Initial state, `S * S * H * n_seqs` elements

Value

A tensor `[S * H, n_tokens + S * n_seqs]`

Shapes

With head size `S`, `H` heads, `n_tokens` tokens and `n_seqs` sequences: `k`, `v`, `r` and `td` are `[S, H, n_tokens]`, `tf` is `[S, H]`, and `state` holds `S * S * H * n_seqs` elements (typically `[S * S * H, n_seqs]`). All must be contiguous.

`n_tokens` counts the tokens of *all* sequences together, so it must be a multiple of `n_seqs`: the kernel derives the per-sequence length as `n_tokens / n_seqs` and restarts the state at each boundary.

Packed result

The result is `[S * H, n_tokens + S * n_seqs]`: the per-token outputs occupy the first `n_tokens` columns and the final state the remaining `S * n_seqs`. `ggml_rwkv_output` and `ggml_rwkv_state` view either half.

Inference only

`GGML_OP_RWKV_WKV6` has no backward pass in ggml.

See Also

ggml_rwkv_wkv7, ggml_rwkv_output

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

---

ggml_rwkv_wkv6_back *Backward Pass of the RWKV-6 WKV Recurrence*

Description

Gradients of ggml_rwkv_wkv6 with respect to all six of its inputs, given grad, the gradient of the loss with respect to the recurrence's packed forward result.

Usage

```
ggml_rwkv_wkv6_back(ctx, k, v, r, tf, td, state, grad)
```

Arguments

ctx	GGML context
k, v, r, tf, td, state	The tensors passed to the forward recurrence
grad	Gradient w.r.t. the packed forward result

Details

Training reaches this automatically through the backward graph; it is exported so the gradient can be checked against a numeric one.

Value

A flat tensor holding the six gradients, concatenated

Packed result

All six gradients come back in one flat tensor, in the order the forward op takes its inputs: [d_k | d_v | d_r | d_tf | d_td | d_state].

See Also

ggml_rwkv_wkv6

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

ggml_rwkv_wkv7 *RWKV-7 WKV Recurrence*

Description

The RWKV-7 ("Goose") recurrence. Where RWKV-6 decays the state by a vector, RWKV-7 applies a rank-1 update built from `a` and `b` alongside the decay `w`, giving the state a richer transition.

Usage

```
ggml_rwkv_wkv7(ctx, r, w, k, v, a, b, state)
```

Arguments

`ctx`	GGML context
`r`	Receptance, `[S, H, n_tokens]`
`w`	Decay, `[S, H, n_tokens]`
`k`	Keys, `[S, H, n_tokens]`
`v`	Values, `[S, H, n_tokens]`
`a`	Removal (state update term), `[S, H, n_tokens]`
`b`	Addition (state update term), `[S, H, n_tokens]`
`state`	Initial state, `S * S * H * n_seqs` elements

Value

A tensor `[S * H, n_tokens + S * n_seqs]`

Shapes

`r`, `w`, `k`, `v`, `a` and `b` are `[S, H, n_tokens]`; `state` holds `S * S * H * n_seqs` elements. All must be contiguous. The result is `[S * H, n_tokens + S * n_seqs]`, outputs then final state – see `ggml_rwkv_output`.

`n_tokens` counts the tokens of *all* sequences together, so it must be a multiple of `n_seqs`.

Inference only

`GGML_OP_RWKV_WKV7` has no backward pass in ggml.

See Also

`ggml_rwkv_wkv6`, `ggml_rwkv_output`

Other state-space: `ggml_gated_linear_attn()`, `ggml_gated_linear_attn_back()`, `ggml_rwkv_output()`, `ggml_rwkv_wkv6()`, `ggml_rwkv_wkv6_back()`, `ggml_rwkv_wkv7_back()`, `ggml_ssm_conv()`, `ggml_ssm_conv_back()`, `ggml_ssm_scan()`, `ggml_ssm_scan_back()`, `ggml_ssm_scan_output()`

ggml_rwkv_wkv7_back *Backward Pass of the RWKV-7 WKV Recurrence*

Description

Gradients of ggml_rwkv_wkv7 with respect to all seven of its inputs, packed in forward source order: [d_r | d_w | d_k | d_v | d_a | d_b | d_state].

Usage

```
ggml_rwkv_wkv7_back(ctx, r, w, k, v, a, b, state, grad)
```

Arguments

ctx	GGML context
r, w, k, v, a, b, state	The tensors passed to the forward recurrence
grad	Gradient w.r.t. the packed forward result

Value

A flat tensor holding the seven gradients, concatenated

See Also

ggml_rwkv_wkv7

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

ggml_save_model *Save a Full Model (Architecture + Weights)*

Description

Saves both the architecture and trained weights of a model to an RDS file. Unlike ggml_save_weights(), which requires the model to be manually reconstructed before loading, ggml_save_model() saves everything needed to restore the model with a single call to ggml_load_model().

Usage

```
ggml_save_model(model, path)
```

Arguments

model	A trained ggml_sequential_model or ggml_functional_model.
path	File path (typically .rds).

Value

The model (invisibly).

Supported model types

- ggml_sequential_model — input shape, layer configs, trained weights, and compilation settings are all saved.
- ggml_functional_model — input/output node graphs (pure R lists, no ggml pointers) and trained node_weights are saved.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(16L, activation = "relu", input_shape = 4L) |>
  ggml_layer_dense(2L,  activation = "softmax")
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")
x <- matrix(runif(64 * 4), 64, 4)
y <- matrix(c(rep(c(1,0), 32), rep(c(0,1), 32)), 64, 2)
model <- ggml_fit(model, x, y, epochs = 1L, batch_size = 32L, verbose = 0L)
tmp <- tempfile(fileext = ".rds")
ggml_save_model(model, tmp)
model2 <- ggml_load_model(tmp)
```

---

ggml_save_weights *Save Model Weights to File*

Description

Saves the trained weights of a sequential model to an RDS file. The file includes both weights and architecture metadata for validation when loading.

Usage

```
ggml_save_weights(model, path)
```

Arguments

model	A trained ggml_sequential_model
path	File path to save weights (typically with .rds extension)

Value

The model (invisibly).

`ggml_scale` *Scale (Graph)*

Description

Creates a graph node for scaling tensor by a scalar: x * s

Usage

```
ggml_scale(ctx, a, s)
```

Arguments

`ctx`	GGML context
`a`	Input tensor
`s`	Scalar value to multiply by

Value

Tensor representing the scaled values

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_scale(ctx, a, 2.0)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [2, 4, 6, 8]
ggml_free(ctx)
```

ggml_scale_bias *Scale and Shift by Constants (Graph)*

Description

Creates a graph node computing `s * a + b` for scalar `s` and `b`. Unlike `ggml_add1()` with `ggml_new_f32()`, this adds a constant without materialising a data-carrying tensor, so it is usable while building a graph in a `no_alloc` context.

Usage

```
ggml_scale_bias(ctx, a, s, b)
```

Arguments

`ctx`	GGML context
`a`	Input tensor
`s`	Scale factor (numeric scalar)
`b`	Bias added after scaling (numeric scalar)

Value

Tensor representing `s * a + b`

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_scale_bias(ctx, a, 2.0, 1.0)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [3, 5, 7, 9]
ggml_free(ctx)
```

ggml_scale_inplace *Scale Tensor In-place (Graph)*

Description

Creates a graph node for in-place scaling: a * s

Usage

```
ggml_scale_inplace(ctx, a, s)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)
s	Scalar value to multiply by

Value

View of tensor a with scaled values

---

ggml_scatter_elements *Scatter elements into a copy of a tensor (ONNX ScatterElements)*

Description

Copies ‘data‘, then writes each element of ‘updates‘ into the copy at the position given by the matching element of ‘indices‘ along ‘axis‘. Every other coordinate is the element’s own, so ‘updates‘ and ‘indices‘ share a shape and only their position along ‘axis‘ is redirected.

Usage

```
ggml_scatter_elements(ctx, data, updates, indices, reduction = 0L, axis = 0L)
```

Arguments

ctx	A ggml context
data	Base tensor (F32); the result starts as a copy of it
updates	Values to write (F32), same shape as ‘indices‘
indices	Destination positions along ‘axis‘ (I32)
reduction	0 to overwrite, 1 to add
axis	Axis to scatter along, in ggml dim order (0 = ne[0])

Value

A new tensor with the shape of ‘data‘

Examples

```
ctx  <- ggml_init(16 * 1024 * 1024)
base <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 4)   # 4 rows of 3
ggml_set_f32(base, rep(0, 12))
upd  <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 2)
ggml_set_f32(upd, c(1, 2, 3, 4, 5, 6))
idx  <- ggml_new_tensor_2d(ctx, GGML_TYPE_I32, 3, 2)
ggml_set_i32(idx, c(3L, 3L, 3L, 0L, 0L, 0L))           # rows 3 and 0
out  <- ggml_scatter_elements(ctx, base, upd, idx, 0L, 1L)
ggml_free(ctx)
```

ggml_schedule_cosine_decay

Cosine annealing LR scheduler

Description

Anneals LR from initial value to ‘eta_min‘ following a cosine curve.

Usage

```
ggml_schedule_cosine_decay(eta_min = 0, T_max = NULL)
```

Arguments

eta_min	Minimum LR at end of schedule
T_max	Total number of epochs (defaults to nepoch from fit state)

Value

List with on_epoch_begin function

See Also

Other callbacks: ggml_callback_early_stopping(), ggml_schedule_reduce_on_plateau(), ggml_schedule_step_decay()

ggml_schedule_reduce_on_plateau

Reduce on plateau LR scheduler

Description

Reduces LR when a metric stops improving.

Usage

```
ggml_schedule_reduce_on_plateau(
  monitor = "val_loss",
  factor = 0.5,
  patience = 5,
  min_lr = 1e-07,
  min_delta = 1e-04,
  mode = "auto"
)
```

Arguments

monitor	Metric to monitor: "val_loss", "train_loss", etc.
factor	Factor to reduce LR by
patience	Epochs with no improvement before reducing
min_lr	Minimum LR
min_delta	Minimum change to qualify as improvement
mode	"min" or "max". "auto" infers from monitor name.

Value

List with on_epoch_end function

See Also

Other callbacks: ggml_callback_early_stopping(), ggml_schedule_cosine_decay(), ggml_schedule_step_decay()

---

ggml_schedule_step_decay *Step decay LR scheduler*

Description

Reduces LR by a factor every ‘step_size‘ epochs.

Usage

```
ggml_schedule_step_decay(step_size = 10, gamma = 0.1)
```

Arguments

step_size	Reduce LR every this many epochs
gamma	Multiplicative factor of LR reduction

Value

List with on_epoch_begin function

See Also

Other callbacks: ggml_callback_early_stopping(), ggml_schedule_cosine_decay(), ggml_schedule_reduce_on_pl

ggml_set *Set Tensor Region (Graph)*

Description

Copies tensor b into tensor a at a specified offset. This allows writing to a portion of a tensor.

Usage

```
ggml_set(ctx, a, b, nb1, nb2, nb3, offset)
```

Arguments

ctx	GGML context
a	Destination tensor
b	Source tensor (data to copy)
nb1	Stride for dimension 1 (in bytes)
nb2	Stride for dimension 2 (in bytes)
nb3	Stride for dimension 3 (in bytes)
offset	Byte offset in destination tensor

Value

Tensor representing the set operation

ggml_set_1d *Set 1D Tensor Region (Graph)*

Description

Simplified 1D version of ggml_set. Copies tensor b into tensor a starting at offset.

Usage

```
ggml_set_1d(ctx, a, b, offset)
```

Arguments

ctx	GGML context
a	Destination tensor
b	Source tensor
offset	Byte offset in destination tensor

Value

Tensor representing the set operation

ggml_set_2d *Set 2D Tensor Region (Graph)*

Description

Simplified 2D version of ggml_set.

Usage

```
ggml_set_2d(ctx, a, b, nb1, offset)
```

Arguments

ctx	GGML context
a	Destination tensor
b	Source tensor
nb1	Stride for dimension 1 (in bytes)
offset	Byte offset in destination tensor

Value

Tensor representing the set operation

ggml_set_abort_callback_default *Restore Default Abort Behavior*

Description

Restores GGML to default abort behavior (prints to stderr and aborts).

Usage

```
ggml_set_abort_callback_default()
```

Value

NULL invisibly

See Also

Other logging: ggml_abort_is_r_enabled(), ggml_log_is_r_enabled(), ggml_log_set_default(), ggml_log_set_r(), ggml_set_abort_callback_r()

ggml_set_abort_callback_r *Enable R-compatible Abort Handling*

Description

Converts GGML abort calls into R errors (via Rf_error). This allows R to catch GGML failures with tryCatch.

Usage

```
ggml_set_abort_callback_r()
```

Value

NULL invisibly

See Also

Other logging: ggml_abort_is_r_enabled(), ggml_log_is_r_enabled(), ggml_log_set_default(), ggml_log_set_r(), ggml_set_abort_callback_default()

Examples

```
ggml_set_abort_callback_r()
# Now GGML aborts will become R errors
result <- tryCatch({
  # ... ggml operations that might fail ...
}, error = function(e) {
  message("GGML error caught: ", e$message)
})
```

ggml_set_f32 *Set F32 data*

Description

Set F32 data

Set F32 Data

Usage

```
ggml_set_f32(tensor, data)

ggml_set_f32(tensor, data)
```

Arguments

tensor	Tensor
data	Numeric vector

Value

NULL (invisible)

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
tensor <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(tensor, c(1, 2, 3, 4, 5))
ggml_get_f32(tensor)
ggml_free(ctx)


ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(t, c(1, 2, 3, 4, 5))
ggml_get_f32(t)
ggml_free(ctx)
```

---

ggml_set_f32_nd *Set Single Float Value by N-D Index*

Description

Sets a single f32 value in the tensor at position [i0, i1, i2, i3]. This is a direct data write, not a graph operation.

Usage

```
ggml_set_f32_nd(tensor, i0, i1 = 0, i2 = 0, i3 = 0, value)
```

Arguments

tensor	Tensor pointer
i0, i1, i2, i3	Indices (0-based)
value	Float value to set

Value

NULL (invisible)

ggml_set_i32 *Set I32 Data*

Description

Sets integer data in an I32 tensor. Used for indices (ggml_get_rows) and position tensors (ggml_rope).

Usage

```
ggml_set_i32(tensor, data)
```

Arguments

tensor	Tensor of type GGML_TYPE_I32
data	Integer vector

Value

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
pos <- ggml_new_tensor_1d(ctx, GGML_TYPE_I32, 10)
ggml_set_i32(pos, 0:9)
ggml_get_i32(pos)
ggml_free(ctx)
```

ggml_set_i32_nd *Set Single Int32 Value by N-D Index*

Description

Sets a single i32 value in the tensor at position [i0, i1, i2, i3].

Usage

```
ggml_set_i32_nd(tensor, i0, i1 = 0, i2 = 0, i3 = 0, value)
```

Arguments

tensor	Tensor pointer
i0, i1, i2, i3	Indices (0-based)
value	Integer value to set

Value

NULL (invisible)

---

ggml_set_input *Mark Tensor as Input*

Description

Mark Tensor as Input

Usage

```
ggml_set_input(tensor)
```

Arguments

tensor	Tensor pointer

Value

The tensor (for chaining)

---

ggml_set_loss *Mark a Tensor as the Loss*

Description

Marks a tensor as the quantity to differentiate. ggml_build_backward_expand requires the graph to contain one – together with at least one ggml_set_param tensor, these are the two ends of the backward pass.

Usage

```
ggml_set_loss(tensor)
```

Arguments

tensor	Tensor pointer, usually a scalar produced by ggml_sum or a loss op

Details

A graph missing either is rejected with an error rather than aborting inside ggml.

Value

The tensor (for chaining)

See Also

ggml_set_param, ggml_build_forward_expand_grads

---

ggml_set_n_threads *Set Number of Threads*

Description

Set the number of threads for GGML operations

Usage

```
ggml_set_n_threads(n_threads)
```

Arguments

n_threads	Number of threads to use

Value

Number of threads set

Examples

```
# Use 4 threads
ggml_set_n_threads(4)

# Use all available cores
ggml_set_n_threads(parallel::detectCores())
```

---

ggml_set_name *Set Tensor Name*

Description

Assigns a name to a tensor. Useful for debugging and graph visualization.

Usage

```
ggml_set_name(tensor, name)
```

Arguments

tensor	Tensor pointer
name	Character string name

Value

The tensor (for chaining)

---

ggml_set_no_alloc *Set No Allocation Mode*

Description

When enabled, tensor creation will not allocate memory for data. Useful for creating computation graphs without allocating storage.

Usage

```
ggml_set_no_alloc(ctx, no_alloc)
```

Arguments

ctx	GGML context
no_alloc	Logical, TRUE to disable allocation

Value

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
ggml_set_no_alloc(ctx, TRUE)
ggml_get_no_alloc(ctx)
ggml_set_no_alloc(ctx, FALSE)
ggml_free(ctx)
```

---

ggml_set_omp_threads *Set OpenMP Thread Count*

Description

Directly calls omp_set_num_threads() to limit OpenMP parallelism. Useful in tests to comply with CRAN policy on core usage.

Usage

```
ggml_set_omp_threads(n)
```

Arguments

n	Number of threads

Value

NULL invisibly

---

ggml_set_op_params *Set Tensor Operation Parameters*

Description

Sets the raw op_params bytes for a tensor.

Usage

```
ggml_set_op_params(tensor, params)
```

Arguments

tensor	External pointer to tensor
params	Raw vector of parameters (max 64 bytes)

Value

NULL invisibly

See Also

Other tensor: ggml_are_same_layout(), ggml_get_op_params(), ggml_get_op_params_f32(), ggml_get_op_params_i32(), ggml_set_op_params_f32(), ggml_set_op_params_i32()

---

ggml_set_op_params_f32 *Set Float Op Parameter*

Description

Sets a single float value in tensor op_params at given index.

Usage

```
ggml_set_op_params_f32(tensor, index, value)
```

Arguments

tensor	External pointer to tensor
index	0-based index (0-15 for 64-byte op_params)
value	Numeric value to set

Value

NULL invisibly

See Also

Other tensor: ggml_are_same_layout(), ggml_get_op_params(), ggml_get_op_params_f32(), ggml_get_op_params_i32(), ggml_set_op_params(), ggml_set_op_params_i32()

---

ggml_set_op_params_i32

Set Integer Op Parameter

Description

Sets a single int32 value in tensor op_params at given index.

Usage

```
ggml_set_op_params_i32(tensor, index, value)
```

Arguments

tensor	External pointer to tensor
index	0-based index (0-15 for 64-byte op_params)
value	Integer value to set

Value

NULL invisibly

See Also

Other tensor: ggml_are_same_layout(), ggml_get_op_params(), ggml_get_op_params_f32(), ggml_get_op_params_i32(), ggml_set_op_params(), ggml_set_op_params_f32()

ggml_set_output *Mark Tensor as Output*

Description

Mark Tensor as Output

Usage

```
ggml_set_output(tensor)
```

Arguments

tensor	Tensor pointer

Value

The tensor (for chaining)

ggml_set_param *Set Tensor as Trainable Parameter*

Description

Marks a tensor as a trainable parameter for backpropagation. The optimizer will compute gradients for this tensor during training.

Usage

```
ggml_set_param(tensor)
```

Arguments

tensor	Tensor pointer

Value

The tensor (for chaining)

See Also

ggml_set_loss, which marks the other end of the backward pass, and ggml_build_backward_expand.

`ggml_set_seed` *Set the random seed for reproducible ggmlR runs*

Description

Fixes the random seed used by ggmlR for everything that is stochastic:

- weight initialisation (sequential, functional and autograd layers),
- dropout masks (training-time),
- data shuffling in the autograd dataloader / training loops.

Usage

```
ggml_set_seed(seed)
```

Arguments

`seed`	A single integer (or value coercible to integer) used as the RNG seed. `NULL` is a no-op, which lets callers thread an optional seed argument through without special-casing it.

Details

This is a thin wrapper around `set.seed`: all randomness in ggmlR is produced by the base R RNG, so a fixed seed gives identical starting weights, dropout masks and batch ordering across runs. It is the single point of control used by the **mlr3** learners (`seed` hyperparameter) and the **parsnip** `"ggml"` engine (`seed` engine argument).

GPU note: this controls the random *inputs* to the computation, not the floating-point arithmetic itself. GPU (Vulkan) kernels are run-to-run stable on a given device/driver for the standard forward/backward paths, but ggmlR does not guarantee *bit-for-bit* identical results across different devices, drivers or backends (CPU vs Vulkan). Reproducibility is at the level of training dynamics, not exact bits.

Value

Invisibly returns `seed`.

See Also

`set.seed`

Examples

```
ggml_set_seed(42)
a <- runif(3)
ggml_set_seed(42)
b <- runif(3)
identical(a, b)  # TRUE
```

ggml_set_zero *Set Tensor to Zero*

Description

Sets all elements of a tensor to zero. This is more efficient than manually setting all elements.

Usage

```
ggml_set_zero(tensor)
```

Arguments

tensor	Tensor to zero out

Value

NULL (invisible)

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_set_f32(t, 1:10)
ggml_set_zero(t)
ggml_get_f32(t)
ggml_free(ctx)
```

ggml_sgn *Sign Function (Graph)*

Description

Creates a graph node for element-wise sign function. sgn(x) = -1 if x < 0, 0 if x == 0, 1 if x > 0

Usage

```
ggml_sgn(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the sign operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -0.5, 0, 0.5, 2))
r <- ggml_sgn(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # c(-1, -1, 0, 1, 1)
ggml_free(ctx)
```

---

ggml_sigmoid *Sigmoid Activation (Graph)*

Description

Creates a graph node for sigmoid activation: 1 / (1 + exp(-x))

Usage

```
ggml_sigmoid(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the sigmoid operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
result <- ggml_sigmoid(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

ggml_sigmoid_inplace *Sigmoid Activation In-place (Graph)*

Description

Creates a graph node for in-place sigmoid activation: 1 / (1 + e^(-x))

Usage

```
ggml_sigmoid_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with sigmoid applied

ggml_silu *SiLU Activation (Graph)*

Description

Creates a graph node for SiLU (Sigmoid Linear Unit) activation, also known as Swish. CRITICAL for LLaMA models.

Usage

```
ggml_silu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the SiLU operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
result <- ggml_silu(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

---

ggml_silu_back *SiLU Backward (Graph)*

Description

Computes the backward pass for SiLU (Swish) activation. Used during training for gradient computation.

Usage

```
ggml_silu_back(ctx, a, b)
```

Arguments

ctx	GGML context
a	Forward input tensor
b	Gradient tensor from upstream

Value

Gradient tensor for the input

---

ggml_silu_inplace *SiLU Activation In-place (Graph)*

Description

Creates a graph node for in-place SiLU (Sigmoid Linear Unit) activation. CRITICAL for LLaMA models with memory efficiency.

Usage

```
ggml_silu_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with SiLU applied

---

ggml_sin *Sine (Graph)*

Description

Creates a graph node for element-wise sine: sin(x)

Usage

```
ggml_sin(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the sin operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(0, pi/6, pi/2, pi))
result <- ggml_sin(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [0, 0.5, 1, 0]
ggml_free(ctx)
```

ggml_snapshot_free *Release a Snapshot Made by a Trainer*

Description

Frees the weights buffer and scheduler of a model returned by a ggml_trainer's $snapshot(). Calling it twice is harmless. The snapshot must not be used afterwards.

Usage

```
ggml_snapshot_free(model)
```

Arguments

model	A snapshot from $snapshot()

Value

NULL, invisibly.

See Also

ggml_trainer

Other optimization: ggml_fit_opt(), ggml_fit_opt_gen(), ggml_fit_opt_multi(), ggml_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), ggml_opt_dataset_get_batch(), ggml_opt_dataset_get_batch_head(), ggml_opt_dataset_init(), ggml_opt_dataset_labels(), ggml_opt_dataset_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weigh ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_fit_multi(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_init_for_fit_multi(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_i(), ggml_opt_loss_type_binary_cross_entropy(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_huber(), ggml_opt_loss_type_mae(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), ggml_opt_n_loss(), ggml_opt_ncorrect(), ggml_opt_optimizer_name(), ggml_opt_optimizer_type_adamw(), ggml_opt_optimizer_type_sgd(), ggml_opt_outputs(), ggml_opt_outputs_i(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_loss_i(), ggml_opt_result_n_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs(), ggml_trainer()

ggml_soft_max *Softmax (Graph)*

Description

Creates a graph node for softmax operation. CRITICAL for attention mechanisms.

Usage

```
ggml_soft_max(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the softmax operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_soft_max(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)
# Output sums to 1.0
ggml_free(ctx)
```

ggml_soft_max_add_sinks *Attach Attention Sinks to a Softmax Node*

Description

The ggml_soft_max_ext counterpart of ggml_flash_attn_ext_add_sinks.

Usage

```
ggml_soft_max_add_sinks(tensor, sinks)
```

Arguments

tensor	A softmax node
sinks	Sink logits, one per head

Value

The tensor (for chaining)

---

ggml_soft_max_ext *Extended Softmax with Masking and Scaling (Graph)*

Description

Creates a graph node for fused softmax operation with optional masking and ALiBi (Attention with Linear Biases) support. Computes: softmax(a * scale + mask * (ALiBi slope)) CRITICAL for efficient attention computation in transformers.

Usage

```
ggml_soft_max_ext(ctx, a, mask = NULL, scale = 1, max_bias = 0)
```

Arguments

ctx	GGML context
a	Input tensor (typically attention scores)
mask	Optional attention mask tensor (F16 or F32). NULL for no mask. Shape must be broadcastable to input tensor.
scale	Scaling factor, typically 1/sqrt(head_dim)
max_bias	Maximum ALiBi bias (0.0 to disable ALiBi)

Details

This extended softmax is commonly used in transformer attention: 1. Scale attention scores by 1/sqrt(d_k) for numerical stability 2. Apply attention mask (e.g., causal mask, padding mask) 3. Optionally apply ALiBi position bias 4. Compute softmax

All these operations are fused for efficiency.

Value

Tensor representing the scaled and masked softmax

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
scores <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 10)
ggml_set_f32(scores, rnorm(100))
attn <- ggml_soft_max_ext(ctx, scores, NULL, 1.0, max_bias = 0.0)
graph <- ggml_build_forward_expand(ctx, attn)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_soft_max_ext_back

Softmax Backward Extended (Graph)

Description

Backward pass for extended softmax operation.

Usage

```
ggml_soft_max_ext_back(ctx, a, b, scale = 1, max_bias = 0)
```

Arguments

ctx	GGML context
a	Softmax output tensor (from forward pass)
b	Gradient tensor from upstream
scale	Scale factor (same as forward pass)
max_bias	Maximum ALiBi bias (same as forward pass)

Value

Gradient tensor for the input

---

ggml_soft_max_ext_back_inplace

Extended Softmax Backward Inplace (Graph)

Description

Creates a graph node for the backward pass of extended softmax, modifying in place.

Usage

```
ggml_soft_max_ext_back_inplace(ctx, a, b, scale = 1, max_bias = 0)
```

Arguments

ctx	GGML context
a	Gradient tensor from upstream
b	Softmax output from forward pass
scale	Scaling factor used in forward pass
max_bias	Maximum ALiBi bias used in forward pass

Value

View of input tensor with gradient computed in place

See Also

Other softmax: ggml_soft_max_ext_inplace()

---

ggml_soft_max_ext_inplace

Extended Softmax Inplace (Graph)

Description

Creates a graph node for extended softmax, modifying input tensor in place. Returns a view of the input tensor.

Usage

```
ggml_soft_max_ext_inplace(ctx, a, mask = NULL, scale = 1, max_bias = 0)
```

Arguments

ctx	GGML context
a	Input tensor (typically attention scores)
mask	Optional attention mask tensor (F16 or F32). NULL for no mask. Shape must be broadcastable to input tensor.
scale	Scaling factor, typically 1/sqrt(head_dim)
max_bias	Maximum ALiBi bias (0.0 to disable ALiBi)

Value

View of input tensor with softmax applied in place

See Also

Other softmax: ggml_soft_max_ext_back_inplace()

ggml_soft_max_inplace *Softmax In-place (Graph)*

Description

Creates a graph node for in-place softmax operation. Returns a view of the input tensor.

Usage

```
ggml_soft_max_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of input tensor with softmax applied

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_soft_max_inplace(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

ggml_softplus *Softplus Activation (Graph)*

Description

Creates a graph node for Softplus activation. Softplus(x) = log(1 + exp(x)). A smooth approximation of ReLU.

Usage

```
ggml_softplus(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the Softplus operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
r <- ggml_softplus(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)
ggml_free(ctx)
```

---

ggml_softplus_inplace *Softplus Activation In-place (Graph)*

Description

Creates a graph node for in-place softplus activation: log(1 + e^x)

Usage

```
ggml_softplus_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with softplus applied

GGML_SORT_ORDER_ASC *Sort Order Constants*

Description

Sort Order Constants

Usage

```
GGML_SORT_ORDER_ASC

GGML_SORT_ORDER_DESC
```

Format

Integer constants

An object of class `integer` of length 1.

Details

Constants for specifying sort order in argsort operations.

- `GGML_SORT_ORDER_ASC` (0): Ascending order (smallest first)
- `GGML_SORT_ORDER_DESC` (1): Descending order (largest first)

Value

An integer constant representing a sort order

Examples

```
GGML_SORT_ORDER_ASC   # 0 - Ascending order
GGML_SORT_ORDER_DESC  # 1 - Descending order

# Usage with ggml_argsort
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(3, 1, 4, 1, 5))
# Get ascending sort indices
idx_asc <- ggml_argsort(ctx, a, GGML_SORT_ORDER_ASC)
# Get descending sort indices
idx_desc <- ggml_argsort(ctx, a, GGML_SORT_ORDER_DESC)
ggml_free(ctx)
```

---

ggml_sqr *Square (Graph)*

Description

Creates a graph node for element-wise squaring: x^2

Usage

```
ggml_sqr(ctx, a)
```

Arguments

`ctx`	GGML context
`a`	Input tensor

Value

Tensor representing the square operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 2, 3, 4))
result <- ggml_sqr(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [1, 4, 9, 16]
ggml_free(ctx)
```

---

ggml_sqr_inplace *Square In-place (Graph)*

Description

Creates a graph node for in-place element-wise square: x^2

Usage

```
ggml_sqr_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with squared values

---

ggml_sqrt *Square Root (Graph)*

Description

Creates a graph node for element-wise square root: sqrt(x)

Usage

```
ggml_sqrt(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the sqrt operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 4)
ggml_set_f32(a, c(1, 4, 9, 16))
result <- ggml_sqrt(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [1, 2, 3, 4]
ggml_free(ctx)
```

---

ggml_sqrt_inplace *Square Root In-place (Graph)*

Description

Creates a graph node for in-place element-wise square root.

Usage

```
ggml_sqrt_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with square root values

ggml_ssm_conv *State-Space Convolution (Mamba)*

Description

Depthwise causal convolution over a sequence, the convolution branch of a Mamba block. Each of the d_inner channels is convolved with its own d_conv-tap kernel; the leading d_conv - 1 positions of sx are the carried-over context from the previous chunk, which is what makes the op resumable across chunks.

Usage

```
ggml_ssm_conv(ctx, sx, c)
```

Arguments

ctx	GGML context
sx	Input with carried context, [d_conv - 1 + n_t, d_inner, n_s]
c	Convolution kernels, [d_conv, d_inner]

Value

A tensor [d_inner, n_t, n_s]

Shapes

sx is [d_conv - 1 + n_t, d_inner, n_s] and c is [d_conv, d_inner]; the result is [d_inner, n_t, n_s], where n_t is the number of tokens per sequence and n_s the number of sequences. All tensors are F32.

Inference only

GGML_OP_SSM_CONV has no backward pass in ggml, so a graph containing it cannot be trained – see ggml_ssm_scan.

See Also

ggml_ssm_scan

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

ggml_ssm_conv_back *Backward Pass of the State-Space Convolution*

Description

Gradients of ggml_ssm_conv with respect to both of its inputs, given grad, the gradient of the loss with respect to the convolution's output.

Usage

```
ggml_ssm_conv_back(ctx, sx, c, grad)
```

Arguments

ctx	GGML context
sx	The sx passed to the forward convolution
c	The c passed to the forward convolution
grad	Gradient w.r.t. the output, shaped like it: [d_inner, n_t, n_s]

Details

Training reaches this automatically through the backward graph – an ggml_ssm_conv node is differentiable, unlike the other state-space and RWKV ops. It is exported so the gradient can be checked directly against a numeric one.

Value

A flat tensor holding d_sx followed by d_c

Packed result

Both gradients come back in one flat tensor, [d_sx | d_c]: the first ggml_nelements(sx) elements are the gradient with respect to sx, the remaining ggml_nelements(c) the gradient with respect to c. A single op returns a single tensor, and packing them keeps the convolution from being run once per input.

See Also

ggml_ssm_conv

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_scan(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

ggml_ssm_scan *Selective State-Space Scan (Mamba / Mamba-2)*

Description

The recurrent core of a Mamba block: a linear state-space recurrence whose transition is modulated per token by `dt`, `B` and `C`. `ids` selects which state row each sequence continues from, so several sequences can share one state buffer.

Usage

```
ggml_ssm_scan(ctx, s, x, dt, A, B, C, ids)
```

Arguments

`ctx`	GGML context
`s`	Initial states, `[d_state, head_dim, n_head, n_seqs_total]`
`x`	Inputs, `[head_dim, n_head, n_seq_tokens, n_seqs]`
`dt`	Per-token timestep, `[n_head, n_seq_tokens, n_seqs]`
`A`	State decay, `[1 or d_state, n_head]`
`B`	Input projection, `[d_state, n_group, n_seq_tokens, n_seqs]`
`C`	Output projection, same shape as `B`
`ids`	Sequence-to-state mapping, `[n_seqs]` I32, 0-based

Value

A flat tensor: outputs followed by final states

Shapes

With `d_state` the state width, `head_dim` the per-head channel count, `n_head` heads, `n_seq_tokens` tokens and `n_seqs` sequences:

- `s` — `[d_state, head_dim, n_head, n_seqs_total]`, contiguous
- `x` — `[head_dim, n_head, n_seq_tokens, n_seqs]`
- `dt` — `[n_head, n_seq_tokens, n_seqs]`, contiguous
- `A` — `[1, n_head]` (Mamba-2) or `[d_state, n_head]` (Mamba-1)
- `B`, `C` — `[d_state, n_group, n_seq_tokens, n_seqs]`, same shape
- `ids` — `[n_seqs]`, type `GGML_TYPE_I32`, 0-based

Packed result

The result is a *flat* tensor holding the outputs followed by the final states: `ggml_nelements(x)` output elements, then `d_state * head_dim * n_head * n_seqs` state elements. Use `ggml_ssm_scan_output` and `ggml_ssm_scan_state` to view either half rather than computing the offsets by hand – getting them wrong silently reads the state as output.

Inference only

GGML_OP_SSM_SCAN has no backward pass in ggml. A graph containing it runs forward on CPU and Vulkan, but cannot be trained: ggml_fit() and the autograd engine will not produce gradients for it.

See Also

ggml_ssm_scan_output, ggml_ssm_scan_state, ggml_ssm_conv

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan_back(), ggml_ssm_scan_output()

---

ggml_ssm_scan_back *Backward Pass of the Selective State-Space Scan*

Description

Gradients of ggml_ssm_scan with respect to all six of its differentiable inputs, given grad, the gradient of the loss with respect to the scan's packed forward result.

Usage

```
ggml_ssm_scan_back(ctx, s, x, dt, A, B, C, ids, grad)
```

Arguments

ctx	GGML context
s	Initial states, as passed to the forward scan
x	Inputs, as passed to the forward scan
dt	Per-token timestep, as passed to the forward scan
A	State decay, as passed to the forward scan
B	Input projection, as passed to the forward scan
C	Output projection, as passed to the forward scan
ids	Sequence-to-state mapping, as passed to the forward scan
grad	Gradient w.r.t. the packed forward result

Details

Training reaches this automatically through the backward graph. It is exported so the gradient can be checked directly against a numeric one.

Value

A flat tensor holding the six gradients, concatenated

Packed result

All six gradients come back in one flat tensor, in the same order the scan takes its inputs: `[d_s | d_x | d_dt | d_A | d_B | d_C]`. `ids` is integer and has no gradient. A single op returns a single tensor, and packing them keeps the recurrence from being replayed once per input.

Cost

The forward pass does not keep its intermediate states, so this replays the recurrence before walking it back – roughly twice the forward work, and no extra memory beyond one state buffer.

See Also

ggml_ssm_scan

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_output()

---

ggml_ssm_scan_output *Views Into a Packed ssm_scan Result*

Description

ggml_ssm_scan returns the outputs and the final states in one flat tensor. These helpers view either half with the right shape, so the offset arithmetic lives in one place: `ggml_ssm_scan_output()` yields the outputs shaped like `x`, and `ggml_ssm_scan_state()` the final states shaped like the leading `n_seqs` rows of `s`.

Usage

```
ggml_ssm_scan_output(ctx, result, x)

ggml_ssm_scan_state(ctx, result, s, ids)
```

Arguments

`ctx`	GGML context
`result`	The tensor returned by ggml_ssm_scan
`x`	The `x` passed to the scan (gives the output shape)
`s`	The `s` passed to the scan (gives the state shape)
`ids`	The `ids` passed to the scan (gives the sequence count)

Details

Both are graph views sharing memory with `result`; they add no computation. `x`, `s` and `ids` must be the same tensors the scan was built with, since their shapes are what the offsets are derived from.

Value

For ggml_ssm_scan_output(), a tensor shaped like x

For ggml_ssm_scan_state(), a tensor [d_state, head_dim, n_head, n_seqs]

See Also

Other state-space: ggml_gated_linear_attn(), ggml_gated_linear_attn_back(), ggml_rwkv_output(), ggml_rwkv_wkv6(), ggml_rwkv_wkv6_back(), ggml_rwkv_wkv7(), ggml_rwkv_wkv7_back(), ggml_ssm_conv(), ggml_ssm_conv_back(), ggml_ssm_scan(), ggml_ssm_scan_back()

---

ggml_step *Step Function (Graph)*

Description

Creates a graph node for element-wise step function. step(x) = 0 if x <= 0, 1 if x > 0 Also known as the Heaviside step function.

Usage

```
ggml_step(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the step operation

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -0.5, 0, 0.5, 2))
r <- ggml_step(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # c(0, 0, 0, 1, 1)
ggml_free(ctx)
```

ggml_sub *Element-wise Subtraction (Graph)*

Description

Creates a graph node for element-wise subtraction.

Usage

```
ggml_sub(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor
b	Second tensor (same shape as a)

Value

Tensor representing the subtraction operation (a - b)

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
b <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(5, 4, 3, 2, 1))
ggml_set_f32(b, c(1, 1, 1, 1, 1))
result <- ggml_sub(ctx, a, b)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

ggml_sub_inplace *Element-wise Subtraction In-place (Graph)*

Description

Creates a graph node for in-place element-wise subtraction. Result is stored in tensor a, saving memory allocation.

Usage

```
ggml_sub_inplace(ctx, a, b)
```

Arguments

ctx	GGML context
a	First tensor (will be modified in-place)
b	Second tensor (same shape as a)

Value

View of tensor a with the subtraction result

---

ggml_sum *Sum (Graph)*

Description

Creates a graph node that computes the sum of all elements.

Usage

```
ggml_sum(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Scalar tensor with the sum

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(1, 2, 3, 4, 5))
result <- ggml_sum(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # 15
ggml_free(ctx)
```

ggml_sum_rows *Sum Rows (Graph)*

Description

Creates a graph node that computes the sum along rows.

Usage

```
ggml_sum_rows(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor with row sums

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 2)
ggml_set_f32(a, c(1, 2, 3, 4, 5, 6))
result <- ggml_sum_rows(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
output <- ggml_get_f32(result)  # [6, 15]
ggml_free(ctx)
```

ggml_swiglu *SwiGLU (Swish/SiLU Gated Linear Unit) (Graph)*

Description

Creates a graph node for SwiGLU operation. SwiGLU uses SiLU (Swish) as the activation function on the first half. CRITICAL for LLaMA, Mistral, and many modern LLMs.

Usage

```
ggml_swiglu(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (first dimension must be even)

Details

Formula: output = SiLU(x) * gate

Value

Tensor with half the first dimension of input

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 8, 3)
ggml_set_f32(a, rnorm(24))
r <- ggml_swiglu(ctx, a)
graph <- ggml_build_forward_expand(ctx, r)
ggml_graph_compute(ctx, graph)
result <- ggml_get_f32(r)  # Shape: 4x3
ggml_free(ctx)
```

---

ggml_swiglu_split *SwiGLU Split (Graph)*

Description

Creates a graph node for SwiGLU with separate input and gate tensors.

Usage

```
ggml_swiglu_split(ctx, a, b)
```

Arguments

ctx	GGML context
a	Input tensor (the values to be gated)
b	Gate tensor (same shape as a)

Details

Formula: output = SiLU(a) * b

Value

Tensor with same shape as input tensors

ggml_tanh *Tanh Activation (Graph)*

Description

Creates a graph node for hyperbolic tangent activation.

Usage

```
ggml_tanh(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor

Value

Tensor representing the tanh operation

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 5)
ggml_set_f32(a, c(-2, -1, 0, 1, 2))
result <- ggml_tanh(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
ggml_get_f32(result)
ggml_free(ctx)
```

ggml_tanh_inplace *Tanh Activation In-place (Graph)*

Description

Creates a graph node for in-place hyperbolic tangent activation.

Usage

```
ggml_tanh_inplace(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (will be modified in-place)

Value

View of tensor a with tanh applied

---

ggml_task *Construct a single-cell compute task*

Description

A ggml_task is the contract object passed from the extraction layer to the dispatch layer. It bundles the operation name, the dense feature matrix (features in rows, cells in columns — the single-cell convention), the operation parameters and the requested device. It performs no computation.

Usage

```
ggml_task(op, matrix, params = list(), device = c("auto", "vulkan", "cpu"))
```

Arguments

op	Operation name; must be registered in ggml_ops_registry (e.g. "embed").
matrix	A numeric matrix (dense) or dgCMatrix (sparse). Rows are features (genes), columns are cells.
params	Named list of operation parameters (e.g. n_components).
device	"vulkan", "cpu" or "auto" (default).

Value

An object of class ggml_task.

See Also

ggml_run, ggml_ops_registry

---

ggml_tcrossprod *GPU transposed cross product (drop-in for* tcrossprod*)*

Description

Computes A %*% t(B) (or A %*% t(A) when B is NULL) on the Vulkan GPU, returning an R matrix, with a CPU fallback. See ggml_matmul for the precision and dispatch notes.

Usage

```
ggml_tcrossprod(A, B = NULL, device = "auto", prec = "f32")
```

Arguments

A	A numeric matrix.
B	A numeric matrix with ncol(B) == ncol(A), or NULL (default) for A %*% t(A).
device, prec	See ggml_matmul.

Value

A %*% t(B) as a numeric matrix.

---

ggml_tensor_copy *Copy Tensor Data*

Description

Copies raw data from src tensor to dst tensor (must be same size).

Usage

```
ggml_tensor_copy(dst, src)
```

Arguments

dst	Destination tensor
src	Source tensor

Value

NULL (invisible)

---

ggml_tensor_data_ptr *Raw Data Pointer of a Tensor*

Description

Returns the address of the tensor's data buffer as a numeric value. Useful for checking whether two tensors alias the same memory – a view and its source share a pointer.

Usage

```
ggml_tensor_data_ptr(tensor)
```

Arguments

tensor	Tensor pointer

Value

The data address, as a double

ggml_tensor_nb *Get Tensor Strides (nb)*

Description

Returns the byte strides for each dimension of the tensor.

Usage

```
ggml_tensor_nb(tensor)
```

Arguments

tensor	Tensor pointer

Value

Numeric vector of 4 stride values (nb0, nb1, nb2, nb3)

ggml_tensor_num *Count Tensors in Context*

Description

Counts the number of tensors allocated in a context.

Usage

```
ggml_tensor_num(ctx)
```

Arguments

ctx	GGML context

Value

Number of tensors

ggml_tensor_overhead *Get Tensor Overhead*

Description

Returns the memory overhead (metadata) for each tensor in bytes

Usage

```
ggml_tensor_overhead()
```

Value

Size in bytes

Examples

```
ggml_tensor_overhead()
```

ggml_tensor_set_f32_scalar *Fill Tensor with Scalar*

Description

Sets all elements of a f32 tensor to a single value.

Usage

```
ggml_tensor_set_f32_scalar(tensor, value)
```

Arguments

tensor	Tensor pointer
value	Float value to fill with

Value

NULL (invisible)

ggml_tensor_shape *Get Tensor Shape*

Description

Returns the shape of a tensor as a numeric vector of 4 elements (ne0, ne1, ne2, ne3)

Usage

```
ggml_tensor_shape(tensor)
```

Arguments

tensor	Tensor pointer

Value

Numeric vector of length 4 with dimensions

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 10, 20)
ggml_tensor_shape(t)
ggml_free(ctx)
```

ggml_tensor_type *Get Tensor Type*

Description

Returns the data type of a tensor as an integer code

Usage

```
ggml_tensor_type(tensor)
```

Arguments

tensor	Tensor pointer

Value

Integer type code (0 = F32, 1 = F16, etc.)

Examples

```
ctx <- ggml_init(1024 * 1024)
t <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
ggml_tensor_type(t)
ggml_free(ctx)
```

---

ggml_test *Test GGML*

Description

Runs GGML library self-test and prints version info.

Usage

```
ggml_test()
```

Value

TRUE if test passed

Examples

```
ggml_test()
```

---

ggml_test_adamw_steps *Compare the AdamW Optimizer Step Across Backends*

Description

Runs `ggml_opt_step_adamw()` for several steps on `backend` and on the CPU from the same starting weights and the same gradients, and reports the weight checksum after each step.

Usage

```
ggml_test_adamw_steps(backend)
```

Arguments

backend	Backend pointer to compare against the CPU.

Details

This needs its own function rather than a case in ggml_test_backend_ops: the op updates the weights and both moment buffers in place, so step N depends on every step before it, and the bias-correction terms it receives change every step. Comparing a single execution proves nothing here – the first Adam step agrees bit-for-bit on both backends and the disagreement only starts at the second.

Value

A data frame with step, w_cpu, w_backend and abs_diff, or a one-column data frame with note when the op could not be run.

See Also

ggml_test_backend_ops

Examples

```
if (ggml_vulkan_available()) {
  be <- ggml_vulkan_init(0)
  print(ggml_test_adamw_steps(be))
  ggml_backend_free(be)
}
```

---

ggml_test_backend_ops *Compare Backend Operations Against the CPU*

Description

Runs each supported operation on backend and on the CPU with identical inputs, and reports the normalized mean squared error between the two. This is the differential test used to find operations whose accelerated implementation disagrees with the reference one.

Usage

```
ggml_test_backend_ops(backend, filter = NULL)
```

Arguments

backend	Backend pointer, e.g. from ggml_vulkan_init(0). Compared against a freshly created CPU backend.
filter	Optional substring; only operations whose name contains it are run. NULL (default) runs all of them.

Details

Each case is executed three times against the same allocated graph and the final run is the one compared, so faults that only appear when a graph is re-executed are visible. That matters: both backend defects found in this package so far were of that kind – the first run was exact and later ones were not.

Value

A data frame with columns `op`, `nmse` and `note`. `nmse` is `NA` when the case could not be run, and `note` then says why.

Examples

```
if (ggml_vulkan_available()) {
  be <- ggml_vulkan_init(0)
  res <- ggml_test_backend_ops(be)
  res[res$nmse > 1e-6, ]        # operations that disagree
  ggml_backend_free(be)
}
```

---

ggml_time_init *Initialize GGML Timer*

Description

Initializes the GGML timing system. Call this once at the beginning of the program before using ggml_time_ms() or ggml_time_us().

Usage

```
ggml_time_init()
```

Value

NULL (invisible)

Examples

```
ggml_time_init()
start <- ggml_time_ms()
Sys.sleep(0.01)
elapsed <- ggml_time_ms() - start
```

ggml_time_ms *Get Time in Milliseconds*

Description

Returns the current time in milliseconds since the timer was initialized.

Usage

```
ggml_time_ms()
```

Value

Numeric value representing milliseconds

Examples

```
ggml_time_init()
start <- ggml_time_ms()
Sys.sleep(0.01)
elapsed <- ggml_time_ms() - start
```

ggml_time_us *Get Time in Microseconds*

Description

Returns the current time in microseconds since the timer was initialized. More precise than ggml_time_ms() for micro-benchmarking.

Usage

```
ggml_time_us()
```

Value

Numeric value representing microseconds

Examples

```
ggml_time_init()
start <- ggml_time_us()
Sys.sleep(0.001)
elapsed <- ggml_time_us() - start
```

ggml_timestep_embedding *Timestep Embedding (Graph Operation)*

Description

Creates sinusoidal timestep embeddings as used in diffusion models. Reference: CompVis/stable-diffusion util.py timestep_embedding

Usage

```
ggml_timestep_embedding(ctx, timesteps, dim, max_period = 10000L)
```

Arguments

ctx	GGML context
timesteps	Input tensor of timestep values [N]
dim	Embedding dimension
max_period	Maximum period for sinusoidal embedding (default 10000)

Value

Tensor of shape [N, dim] with timestep embeddings

ggml_top_k *Top-K Indices (Graph)*

Description

Returns the indices of top K elements per row. Useful for sampling strategies in language models (top-k sampling). Note: the resulting indices are in no particular order within top-k.

Usage

```
ggml_top_k(ctx, a, k)
```

Arguments

ctx	GGML context
a	Input tensor (F32)
k	Number of top elements to return per row

Value

Tensor containing I32 indices of top-k elements (not values)

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
# Logits from model output
logits <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
ggml_set_f32(logits, rnorm(100))
# Get top 5 logits for sampling
top5 <- ggml_top_k(ctx, logits, 5)
graph <- ggml_build_forward_expand(ctx, top5)
ggml_graph_compute(ctx, graph)
ggml_free(ctx)
```

---

ggml_tp_dp_forward *TPxDP hybrid matrix multiply across replicas of Vulkan device groups*

Description

Computes `Y = X %*% t(W)` as a hybrid of tensor parallelism and data parallelism: the weight matrix `W` is replicated across `replicas` device groups (data parallelism over the batch `X`), and within each group the weight rows are split across that group's GPUs (tensor parallelism, via `ggml_vulkan_split_mul_mat`). Each replica computes a contiguous shard of the batch rows; the shards are concatenated back into the full result.

Usage

```
ggml_tp_dp_forward(
  W,
  X,
  replicas,
  weights = NULL,
  transport = c("host-staging", "opaque-fd", "device-group")
)
```

Arguments

`W`	Weight matrix, `N` x `K` (replicated to every group).
`X`	Activation matrix, `M` x `K`; its rows (the batch) are split across the replicas.
`replicas`	A list of integer vectors, each a group of physical GPU indices (0-based) forming one replica, e.g. `list(c(0, 1), c(2, 3))`. Within a group the weights are tensor-split; across groups the batch is data-split.
`weights`	Optional numeric vector (length = group size) giving the per-device TP row weighting inside each group (applied to every group). `NULL` (default) splits evenly.
`transport`	Cross-device TP gather transport (see `ggml_vulkan_split_mul_mat`).

Details

This mirrors the inference layout for the 4x P100 target (2 replicas x TP=2): there is no cross-replica communication, so throughput scales with the replica count while each replica's cross-device TP gather stays local to its group.

Value

The `M` x `N` result matrix, equal to `X %*% t(W)` up to the GPU's floating-point accumulation.

See Also

`ggml_vulkan_split_mul_mat` for the single-group TP path.

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 4) {
  W <- matrix(rnorm(2048 * 64), nrow = 2048)
  X <- matrix(rnorm(8 * 64), nrow = 8)      # batch of 8
  # 2 replicas x TP=2: batch split 4/4, weights split within {0,1} and {2,3}
  Y <- ggml_tp_dp_forward(W, X, replicas = list(c(0, 1), c(2, 3)))
  max(abs(Y - X %*% t(W)))
}
```

---

ggml_trainer *A Training Context You Step Yourself*

Description

Builds the graph and optimizer context for a compiled model and hands them back, so a training loop can be written in R rather than handed to `ggml_fit`. The context lives until `$free()`, keeping weights, Adam moments and the compute graph across as many steps as the loop takes.

Usage

```
ggml_trainer(
  model,
  batch_size = 32L,
  nbatch_logical = NULL,
  loss_mask = NULL,
  weight_decay = NULL
)
```

Arguments

model	A compiled ggml_sequential_model, or a compiled functional model with one input and one output
batch_size	Samples per step. The graph is built for exactly this many; every batch handed to $step() must match.
nbatch_logical	Logical batch size for gradient accumulation. Defaults to batch_size, meaning every step updates the weights.
loss_mask	NULL (default) for the compiled loss unweighted; 1 for one weight per sample; the number of outputs for one weight per output element. See the section on loss weights.
weight_decay	Decoupled weight decay in [0, 1]. NULL (default) takes the value given to ggml_compile. Frozen weights are not decayed.

Details

This is the level below ggml_fit_opt_gen. Reach for it when the loop is not a pass over data: reinforcement learning, where experience comes from a replay buffer the agent is still filling and a target network is refreshed every so many steps; curriculum schemes that change the data distribution as they go; data generated fresh for every step; or anything where "epoch" is not the unit of progress.

Do not emulate this by calling ggml_fit once per batch: each such call builds a new optimizer context, so the Adam moments are reset and every step is the optimizer's first.

Value

An object of class ggml_trainer: a list of functions $step(x, y, sample_weight = NULL), $eval(x, y, sample_weight = NULL), $predict(x, batch_size = NULL, matmul_precision = NULL), $snapshot(), $update_target(target, tau = 1), $model(), $calibrate_bn(x), $set_lr(lr), $n_steps(), $free().

Using it

```
tr <- ggml_trainer(model, batch_size = 32L)
on.exit(tr$free())

for (i in seq_len(10000L)) {
  b <- sample_batch(replay, 32L)     # your own bookkeeping
  loss <- tr$step(b$x, b$y)
  if (i %% 500L == 0L) target <- tr$model()   # refresh a target network
}

model <- tr$model()                  # weights back into an R model object
tr$free()
```

What a step does

`$step(x, y, sample_weight)` runs one forward and backward pass and returns that step's loss. Gradient accumulation, if `nbatch_logical` asked for it, is tracked inside the context: the optimizer updates the weights only on the period boundary, and the intervening steps only accumulate. `$eval(x, y, sample_weight)` is the same without the backward pass, for measuring on held-out data.

Loss weights and masks

With `loss_mask` set, the loss is weighted mean squared error, `sum(w * (pred - y)^2) / (batch * outputs)`, and `$step()` and `$eval()` take a `sample_weight` matrix:

- `loss_mask = 1`: `sample_weight` is `[batch, 1]` (or a vector of length `batch`) – one weight per sample, as in Keras.
- `loss_mask = <outputs>`: `sample_weight` is `[batch, outputs]` – one weight per output element. This is *not* Keras' `sample_weight`: a 0/1 matrix is a mask that trains only the outputs whose label is known (e.g. one action of a Q-network), and a 0 removes that element's gradient exactly. The loss is not renormalised by the number of active elements.

Omitting `sample_weight` on a weighted trainer means all ones. Only `loss = "mse"` supports weights.

Models

Sequential models and functional models (`ggml_model`) with **one input and one output** are supported. Functional models with several inputs or outputs are refused. For a functional model with dropout the dropout masks are redrawn before every `$step()`.

Predicting during training

`$predict(x, batch_size)` returns predictions from the current weights, as `ggml_predict` would, in eval mode (no dropout). It runs on a scheduler of its own, created on first use: the weights are copied into a separate buffer for the call and released after it, so neither the training graph's buffers nor the optimizer state are touched. `batch_size` defaults to the trainer's; `x` may have any number of rows.

Target networks

`$snapshot()` returns an independent copy of the model: its own weight tensors in its own buffer and its own scheduler, sharing nothing mutable with the trainer. Later steps do not move it, `ggml_predict` on it never touches the training path, and it outlives `$free()`. It is built in eval mode, so dropout is off in it. Release it with `ggml_snapshot_free` (a finaliser is only a backstop); after that `ggml_predict` and `$update_target` on it are refused with an error.

`$update_target(target, tau = 1)` moves a snapshot towards the current weights, $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$: `tau = 1` is the hard copy of DQN, a small `tau` the Polyak average of DDPG/SAC. Non-trainable buffers (a sequential batch-norm's running mean and variance) are copied whole, never averaged. The target must have the same weights, by name and shape, or it is refused. The update goes through host memory, one read and one write per tensor.

Lifetime

The context holds GPU buffers and must be released with `$free()`; calling it twice is harmless. A trainer that goes out of scope without it is cleaned up when garbage collected, but that moment is not predictable, so free it yourself. After `$free()` every other method errors rather than touching released memory.

The model object passed in is unchanged – R semantics being what they are, the trainer works on its own copy of the weights. `$model()` returns a model carrying the current ones.

The returned model shares the trainer's backend scheduler, and predicting with it re-lays the scheduler's buffers for another graph. The trainer therefore re-allocates its own graph on the first step after any `$model()` call; weights and Adam moments are kept. `$predict()` needs none of this. Predicting with the *original* model object while the trainer is alive is not covered by this and can crash the session on the next step – take the model from `$model()` instead.

Batch norm

A step normalizes each batch by its own statistics and does not update the running ones a sequential `batch_norm` predicts with; `ggml_fit()` recalibrates them after training, a trainer does not. Call `$calibrate_bn(x)` with representative inputs (up to 1024 rows are used) before predicting, snapshotting or saving – or fix the statistics with `ggml_freeze_bn_stats`. Creating a trainer for a model with unfixed batch-norm layers warns once.

See Also

`ggml_fit_opt_gen` for a generator-driven epoch loop, `ggml_fit` for the ordinary in-memory path.

Other optimization: `ggml_fit_opt()`, `ggml_fit_opt_gen()`, `ggml_fit_opt_multi()`, `ggml_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `ggml_opt_dataset_get_batch_head()`, `ggml_opt_dataset_init()`, `ggml_opt_dataset_labels()`, `ggml_opt_dataset_ndata()`, `ggml_opt_dataset_shuffle()`, `ggml_opt_dataset_weigh`
`ggml_opt_default_params()`, `ggml_opt_epoch()`, `ggml_opt_eval()`, `ggml_opt_fit()`, `ggml_opt_fit_multi()`, `ggml_opt_free()`, `ggml_opt_get_lr()`, `ggml_opt_grad_acc()`, `ggml_opt_init()`, `ggml_opt_init_for_fit()`, `ggml_opt_init_for_fit_multi()`, `ggml_opt_inputs()`, `ggml_opt_labels()`, `ggml_opt_loss()`, `ggml_opt_loss_i()`, `ggml_opt_loss_type_binary_cross_entropy()`, `ggml_opt_loss_type_cross_entropy()`, `ggml_opt_loss_type_huber()`, `ggml_opt_loss_type_mae()`, `ggml_opt_loss_type_mean()`, `ggml_opt_loss_type_mse`
`ggml_opt_loss_type_sum()`, `ggml_opt_loss_type_weighted_mse()`, `ggml_opt_n_loss()`, `ggml_opt_ncorrect()`, `ggml_opt_optimizer_name()`, `ggml_opt_optimizer_type_adamw()`, `ggml_opt_optimizer_type_sgd()`, `ggml_opt_outputs()`, `ggml_opt_outputs_i()`, `ggml_opt_pred()`, `ggml_opt_prepare_alloc()`, `ggml_opt_reset()`, `ggml_opt_result_accuracy()`, `ggml_opt_result_free()`, `ggml_opt_result_init()`, `ggml_opt_result_loss()`, `ggml_opt_result_loss_i()`, `ggml_opt_result_n_loss()`, `ggml_opt_result_ndata()`, `ggml_opt_result_pred()`, `ggml_opt_result_reset()`, `ggml_opt_set_lr()`, `ggml_opt_static_graphs()`, `ggml_snapshot_free()`

Examples

```
ggml_set_n_threads(1L)  # deterministic, single OpenMP pool
n <- 64
x <- matrix(runif(n * 4), nrow = n, ncol = 4)
y <- matrix(0, nrow = n, ncol = 2)
for (i in seq_len(n)) { y[i, if (sum(x[i, ]) > 2) 1L else 2L] <- 1 }
```

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(8, activation = "relu") |>
  ggml_layer_dense(2, activation = "softmax")
model$input_shape <- 4L
model <- ggml_compile(model, optimizer = "adam",
                      loss = "categorical_crossentropy")

tr <- ggml_trainer(model, batch_size = 16L)
for (i in 1:4) {
  idx <- ((i - 1L) * 16L + 1L):(i * 16L)
  tr$step(x[idx, , drop = FALSE], y[idx, , drop = FALSE])
}
model <- tr$model()
tr$free()
```

`ggml_training_history` *Training history of a fitted ggml model*

Description

Returns the per-epoch loss / accuracy curve recorded during `ggml_fit`, in a tidy data frame. This is the standard accessor for the loss curve; it works on a raw sequential/functional model or on a fitted parsnip engine object (e.g. from `extract_fit_engine()`).

Usage

```
ggml_training_history(object, format = c("wide", "long"), ...)
```

Arguments

`object`	A fitted `ggml_sequential_model`, `ggml_functional_model` or `ggmlr_parsnip_model`.
`format`	`"wide"` (default) returns one row per epoch with one column per metric. `"long"` returns one row per (epoch, metric) with columns `epoch`, `metric` (e.g. `"loss"`/`"accuracy"`), `split` (`"train"`/`"val"`) and `value` — convenient for **ggplot2** faceting.
`...`	Unused; for extensibility.

Value

A data frame (tibble if **tibble** is installed). Wide columns: `epoch`, `train_loss`, `train_accuracy`, and `val_loss` / `val_accuracy` when a validation split was used. Returns `NULL` with a warning if the model has no recorded history (e.g. not yet fitted).

See Also

`ggml_model_backend`

ggml_transpose *Transpose (Graph)*

Description

Creates a graph node for matrix transpose operation.

Usage

```
ggml_transpose(ctx, a)
```

Arguments

ctx	GGML context
a	Input tensor (2D matrix)

Value

Tensor representing the transposed matrix

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 3, 2)
ggml_set_f32(a, 1:6)
result <- ggml_transpose(ctx, a)
graph <- ggml_build_forward_expand(ctx, result)
ggml_graph_compute(ctx, graph)
shape <- ggml_tensor_shape(result)  # [2, 3]
ggml_free(ctx)
```

GGML_TYPE_F32 *GGML Data Types*

Description

Constants representing different data types supported by GGML.

Usage

```
GGML_TYPE_F32

GGML_TYPE_F16

GGML_TYPE_Q4_0

GGML_TYPE_Q4_1

GGML_TYPE_Q8_0

GGML_TYPE_Q2_K

GGML_TYPE_Q3_K

GGML_TYPE_Q4_K

GGML_TYPE_Q5_K

GGML_TYPE_Q6_K

GGML_TYPE_I32

GGML_TYPE_BF16
```

Format

Integer constants

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

An object of class `integer` of length 1.

Details

- `GGML_TYPE_F32`: 32-bit floating point (default)
- `GGML_TYPE_F16`: 16-bit floating point (half precision)

- GGML_TYPE_Q4_0: 4-bit quantization type 0
- GGML_TYPE_Q4_1: 4-bit quantization type 1
- GGML_TYPE_Q8_0: 8-bit quantization type 0
- GGML_TYPE_I32: 32-bit integer
- GGML_TYPE_BF16: 16-bit brain float (bfloat16)

Value

An integer constant representing a GGML data type

Examples

```
GGML_TYPE_F32
GGML_TYPE_F16
GGML_TYPE_I32
```

---

ggml_type_name *Get Type Name*

Description

Returns the string name of a GGML type.

Usage

```
ggml_type_name(type)
```

Arguments

type	GGML type constant (e.g., GGML_TYPE_F32)

Value

Character string with type name

See Also

Other type_system: ggml_blck_size(), ggml_ftype_to_ggml_type(), ggml_is_quantized(), ggml_type_sizef()

Examples

```
ggml_type_name(GGML_TYPE_F32)  # "f32"
ggml_type_name(GGML_TYPE_Q4_0) # "q4_0"
```

ggml_type_size *Get Type Size in Bytes*

Description

Returns the size in bytes for all elements in a block for a given type.

Usage

```
ggml_type_size(type)
```

Arguments

type	GGML type constant (e.g., GGML_TYPE_F32)

Value

Size in bytes

ggml_type_sizef *Get Type Size as Float*

Description

Returns the size in bytes of a GGML type as a floating-point number. For quantized types, this is the average bytes per element.

Usage

```
ggml_type_sizef(type)
```

Arguments

type	GGML type constant

Value

Numeric size in bytes (can be fractional for quantized types)

See Also

Other type_system: ggml_blck_size(), ggml_ftype_to_ggml_type(), ggml_is_quantized(), ggml_type_name()

Examples

```
ggml_type_sizef(GGML_TYPE_F32)  # 4.0
ggml_type_sizef(GGML_TYPE_F16)  # 2.0
```

ggml_unary_op_name *Get Unary Operation Name*

Description

Returns the string name of a GGML unary operation.

Usage

```
ggml_unary_op_name(op)
```

Arguments

op	GGML unary operation constant

Value

Character string with operation name

See Also

Other op_info: ggml_get_unary_op(), ggml_op_desc(), ggml_op_name(), ggml_op_symbol()

ggml_unfreeze_weights *Unfreeze Layer Weights*

Description

Sets trainable = TRUE on layers, chosen as in ggml_freeze_weights: by layer_names and/or prefix, else by from / to, else all of them.

Usage

```
ggml_unfreeze_weights(
  model,
  from = 1L,
  to = length(model$layers),
  layer_names = NULL,
  prefix = NULL,
  ...
)
```

Arguments

model	A model object (ggml_sequential_model or ggml_functional_model)
from	Integer index of the first layer to unfreeze (default: 1)
to	Integer index of the last layer to unfreeze (default: last layer)
layer_names	Character vector of layer names to unfreeze (overrides from/to)
prefix	Character vector: unfreeze every layer whose name starts with one of these. With layer_names, the union of the two.
...	Additional arguments passed to methods

Value

The model with selected layers unfrozen.

Examples

```
model <- ggml_model_sequential() |>
  ggml_layer_dense(64, activation = "relu") |>
  ggml_layer_dense(10, activation = "softmax")

model <- ggml_freeze_weights(model)
model <- ggml_unfreeze_weights(model, from = 2)  # unfreeze last layer only
```

---

ggml_unmarshal_model *Unmarshal a ggmlR model from an in-memory container*

Description

Reconstructs a ggmlR model previously produced by ggml_marshal_model. Validates the container's format tag, schema version, and (if **digest** is installed) the SHA-256 checksum of the payload before deserializing.

Usage

```
ggml_unmarshal_model(x, backend = NULL)
```

Arguments

x	A "ggmlR_marshaled" container.
backend	Backend selection passed through to ggml_load_model. Default "auto".

Value

A compiled ggmlR model object (sequential or functional).

See Also

ggml_marshal_model, ggml_load_model

ggml_upscale *Upscale Tensor (Graph)*

Description

Upscales tensor by multiplying ne0 and ne1 by scale factor. Supports different interpolation modes for image upscaling.

Usage

```
ggml_upscale(ctx, a, scale_factor, mode = 0L)

GGML_SCALE_MODE_NEAREST

GGML_SCALE_MODE_BILINEAR

GGML_SCALE_MODE_BICUBIC
```

Arguments

ctx	GGML context
a	Input tensor (typically 2D or 4D for images)
scale_factor	Integer scale factor (e.g., 2 = double size)
mode	Scale mode constant (see details)

Format

An object of class integer of length 1.

An object of class integer of length 1.

An object of class integer of length 1.

Details

Scale mode constants:

- GGML_SCALE_MODE_NEAREST (0): Nearest neighbor interpolation - fastest, pixelated
- GGML_SCALE_MODE_BILINEAR (1): Bilinear interpolation - smooth, good balance
- GGML_SCALE_MODE_BICUBIC (2): Bicubic interpolation - smoothest, most compute

Value

Upscaled tensor with dimensions multiplied by scale_factor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
img <- ggml_new_tensor_2d(ctx, GGML_TYPE_F32, 8, 8)
ggml_set_f32(img, rnorm(64))

# Nearest neighbor (fastest, pixelated)
up_nearest <- ggml_upscale(ctx, img, 2, GGML_SCALE_MODE_NEAREST)

# Bilinear (smooth)
up_bilinear <- ggml_upscale(ctx, img, 2, GGML_SCALE_MODE_BILINEAR)

# Bicubic (smoothest)
up_bicubic <- ggml_upscale(ctx, img, 2, GGML_SCALE_MODE_BICUBIC)

graph <- ggml_build_forward_expand(ctx, up_nearest)
ggml_graph_compute(ctx, graph)
# Result is 16x16
ggml_free(ctx)
```

---

ggml_used_mem *Get Used Memory*

Description

Returns the amount of memory currently used in the context

Usage

```
ggml_used_mem(ctx)
```

Arguments

ctx	GGML context

Value

Used memory in bytes

Examples

```
ctx <- ggml_init(1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
ggml_used_mem(ctx)
ggml_free(ctx)
```

ggml_version *Get GGML version*

Description

Get GGML version

Usage

```
ggml_version()
```

Value

Character string with GGML version

Examples

```
ggml_version()
```

ggml_view_1d *1D View with Byte Offset (Graph)*

Description

Creates a 1D view of a tensor starting at a byte offset. The view shares memory with the source tensor.

Usage

```
ggml_view_1d(ctx, a, ne0, offset = 0)
```

Arguments

ctx	GGML context
a	Source tensor
ne0	Number of elements in the view
offset	Byte offset from the start of tensor data

Value

View tensor

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 100)
# View elements 10-19 (offset = 10 * 4 bytes = 40)
v <- ggml_view_1d(ctx, a, 10, 40)
ggml_free(ctx)
```

---

ggml_view_2d *2D View with Byte Offset (Graph)*

Description

Creates a 2D view of a tensor starting at a byte offset. The view shares memory with the source tensor.

Usage

```
ggml_view_2d(ctx, a, ne0, ne1, nb1, offset = 0)
```

Arguments

ctx	GGML context
a	Source tensor
ne0	Size of dimension 0
ne1	Size of dimension 1
nb1	Stride for dimension 1 (in bytes)
offset	Byte offset from the start of tensor data

Value

View tensor

---

ggml_view_3d *3D View with Byte Offset (Graph)*

Description

Creates a 3D view of a tensor starting at a byte offset. The view shares memory with the source tensor.

Usage

```
ggml_view_3d(ctx, a, ne0, ne1, ne2, nb1, nb2, offset = 0)
```

Arguments

ctx	GGML context
a	Source tensor
ne0	Size of dimension 0
ne1	Size of dimension 1
ne2	Size of dimension 2
nb1	Stride for dimension 1 (in bytes)
nb2	Stride for dimension 2 (in bytes)
offset	Byte offset from the start of tensor data

Value

View tensor

---

ggml_view_4d *4D View with Byte Offset (Graph)*

Description

Creates a 4D view of a tensor starting at a byte offset. The view shares memory with the source tensor. CRITICAL for KV-cache operations in transformers.

Usage

```
ggml_view_4d(ctx, a, ne0, ne1, ne2, ne3, nb1, nb2, nb3, offset = 0)
```

Arguments

ctx	GGML context
a	Source tensor
ne0	Size of dimension 0
ne1	Size of dimension 1
ne2	Size of dimension 2
ne3	Size of dimension 3
nb1	Stride for dimension 1 (in bytes)
nb2	Stride for dimension 2 (in bytes)
nb3	Stride for dimension 3 (in bytes)
offset	Byte offset from the start of tensor data

Value

View tensor

ggml_view_tensor *View Tensor*

Description

Creates a view of the tensor (shares data, no copy)

Usage

```
ggml_view_tensor(ctx, src)
```

Arguments

ctx	GGML context
src	Source tensor

Value

View tensor (shares data with src)

Examples

```
ctx <- ggml_init(16 * 1024 * 1024)
a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
view <- ggml_view_tensor(ctx, a)
# view shares data with a
ggml_free(ctx)
```

ggml_vulkan_available *Check if Vulkan support is available*

Description

Returns TRUE if the package was compiled with Vulkan support. To enable Vulkan, install libvulkan-dev and glslc, then reinstall ggmlR.

Usage

```
ggml_vulkan_available()
```

Value

Logical indicating if Vulkan is available

Examples

```
ggml_vulkan_available()
```

ggml_vulkan_backend_name *Get Vulkan backend name*

Description

Returns the name of the Vulkan backend (includes device info).

Usage

```
ggml_vulkan_backend_name(backend)
```

Arguments

backend	Vulkan backend pointer

Value

Character string with backend name

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  backend <- ggml_vulkan_init(0)
  print(ggml_vulkan_backend_name(backend))
  ggml_vulkan_free(backend)
}
```

ggml_vulkan_device_caps *Get Vulkan device capabilities*

Description

Returns hardware capabilities for the specified Vulkan device.

Usage

```
ggml_vulkan_device_caps(device = 0L)
```

Arguments

device	Device index (0-based, default 0)

Value

Named list: coopmat_support, coopmat1_fa_support, fp16, subgroup_size, subgroup_no_shmem

ggml_vulkan_device_count *Get number of Vulkan devices*

Description

Returns the number of available Vulkan-capable GPU devices.

Usage

```
ggml_vulkan_device_count()
```

Value

Integer count of Vulkan devices (0 if Vulkan not available)

Examples

```
if (ggml_vulkan_available()) {
  ggml_vulkan_device_count()
}
```

ggml_vulkan_device_description *Get Vulkan device description*

Description

Returns a human-readable description of the specified Vulkan device.

Usage

```
ggml_vulkan_device_description(device = 0L)
```

Arguments

device	Device index (0-based)

Value

Character string with device description

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  ggml_vulkan_device_description(0)
}
```

ggml_vulkan_device_groups

Probe Vulkan device groups (NVLink / multi-GPU peer access)

Description

Enumerates Vulkan device groups (`VK_KHR_device_group`, also known as Linked Device Adapter / LDA) and probes whether the driver reports direct peer memory access between the physical GPUs in each group.

Usage

```
ggml_vulkan_device_groups()
```

Details

A device group with more than one device *and* peer copy/generic memory features is the prerequisite for true GPU-to-GPU transfers routed over NVLink (or PCIe peer-to-peer) through a single device-group logical device — as opposed to sharing memory as an opaque fd between independent devices, which the driver may route through host memory.

This is a diagnostic only: it does not create any long-lived device group. On a machine with a single GPU it reports zero multi-device groups. Device groups for compute are effectively an NVIDIA feature; AMD/RADV typically reports only single-device groups.

Value

A named list with `n_groups` (integer, number of device groups reported by the driver) and `report` (character, a human-readable per-group diagnostic including peer memory features). Use `cat()` on `report` to read it.

Examples

```
if (ggml_vulkan_available()) {
  g <- ggml_vulkan_device_groups()
  cat(g$report)
}
```

ggml_vulkan_device_memory

Get Vulkan device memory

Description

Returns free and total memory for the specified Vulkan device.

Usage

```
ggml_vulkan_device_memory(device = 0L)
```

Arguments

device	Device index (0-based)

Value

Named list with 'free' and 'total' memory in bytes

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  mem <- ggml_vulkan_device_memory(0)
  cat("Free:", mem$free / 1e9, "GB\n")
  cat("Total:", mem$total / 1e9, "GB\n")
}
```

---

ggml_vulkan_free *Free Vulkan backend*

Description

Releases resources associated with the Vulkan backend.

Usage

```
ggml_vulkan_free(backend)
```

Arguments

backend	Vulkan backend pointer from ggml_vulkan_init()

Value

NULL (invisible)

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  backend <- ggml_vulkan_init(0)
  ggml_vulkan_free(backend)
}
```

ggml_vulkan_hard_exit_available *Is the Vulkan hard-exit path compiled in?*

Description

Reports whether this build of ggmlR was compiled with `-DGGML_VK_HARD_EXIT`, which is what makes `ggml_vulkan_shutdown(hard = TRUE)` actually call `_exit()`.

Usage

```
ggml_vulkan_hard_exit_available()
```

Details

The hard-exit path is **disabled by default**: CRAN Repository Policy forbids a package from terminating the user's R session, so the released package must not link `_exit()`. Builds from source can opt in with `R CMD INSTALL . --configure-args="--enable-hard-exit"` (on Windows, set `Sys.setenv(GGML_VK_HARD_EXIT = "1")` before installing, because R there ignores `configure.args`).

When it is not compiled in, `ggml_vulkan_shutdown(hard = TRUE)` performs the normal teardown and emits a `warning` rather than silently ignoring the request.

Value

`TRUE` if the hard-exit path is available, otherwise `FALSE`.

See Also

`ggml_vulkan_shutdown`

Examples

```
ggml_vulkan_hard_exit_available()
```

ggml_vulkan_init *Initialize Vulkan backend*

Description

Creates a Vulkan backend for the specified device. The backend must be freed with ggml_vulkan_free() when done.

Usage

```
ggml_vulkan_init(device = 0L)
```

Arguments

device	Device index (0-based, default 0)

Value

Vulkan backend pointer

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  backend <- ggml_vulkan_init(0)
  print(ggml_vulkan_backend_name(backend))
  ggml_vulkan_free(backend)
}
```

---

ggml_vulkan_is_backend

Check if backend is Vulkan

Description

Returns TRUE if the given backend is a Vulkan backend.

Usage

```
ggml_vulkan_is_backend(backend)
```

Arguments

backend	Backend pointer

Value

Logical indicating if backend is Vulkan

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  vk_backend <- ggml_vulkan_init(0)
  cpu_backend <- ggml_backend_cpu_init()

  ggml_vulkan_is_backend(vk_backend)  # TRUE
  ggml_vulkan_is_backend(cpu_backend) # FALSE

  ggml_vulkan_free(vk_backend)
  ggml_backend_free(cpu_backend)
}
```

ggml_vulkan_list_devices

List all Vulkan devices

Description

Returns detailed information about all available Vulkan devices.

Usage

```
ggml_vulkan_list_devices()
```

Value

List of device information (index, name, memory)

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() > 0) {
  devices <- ggml_vulkan_list_devices()
  print(devices)
}
```

ggml_vulkan_p2p_selftest

Opaque-fd device-to-device P2P self-test

Description

Exercises the `VK_KHR_external_memory_fd` transport used by Vulkan tensor parallelism to move data between GPUs. A byte pattern is written on the source device, its memory is exported as an opaque fd and imported on the destination device, copied into a local buffer there, then read back and compared.

Usage

```
ggml_vulkan_p2p_selftest(
  src_device,
  dst_device,
  bytes = 64L * 1024L * 1024L,
  iters = 50L,
  transport = c("host-staging", "opaque-fd", "device-group")
)
```

Arguments

src_device	Source GPU index (0-based).
dst_device	Destination GPU index (0-based). Equal to src_device for a loopback sanity check.
bytes	Transfer size in bytes (default 64 MiB).
iters	Number of copies to time for the bandwidth measurement (default 50).
transport	Cross-device transport to exercise: "host-staging" (default, portable device->host->device copy — correct on every driver, PCIe + RAM bounded), "opaque-fd" (VK_KHR_external_memory_fd P2P; works on AMD/RADV but does NOT alias memory cross-device on the NVIDIA proprietary driver — reads back as zeros), or "device-group" (experimental NVIDIA LDA). host-staging is the transport Stage E3 uses.

Details

When src_device == dst_device the test runs in *loopback* mode (export and import on the same GPU) — a sanity check of the fd mechanism that touches no inter-device link. When the devices differ it runs *cross-device*: after verifying correctness it times iters device-to-device copies and reports the achieved bandwidth.

Interpreting bandwidth: a measured rate above the PCIe 3.0 x16 ceiling (~16 GB/s) is *empirical* evidence that a faster physical link (e.g. NVLink) carried the bytes. It is **not** a claim that Vulkan used an NVLink API — Vulkan exposes no call to query the route, so the conclusion is inferred from the rate alone, never asserted from the API.

Value

A named list: status (integer, 0 = data verified, <0 = failure), gbps (numeric, measured cross-device bandwidth; 0 for loopback or on failure) and report (character diagnostic, incl. the NVLink-vs-PCIe inference).

Examples

```
if (ggml_vulkan_available()) {
  # loopback sanity check on device 0
  r <- ggml_vulkan_p2p_selftest(0L, 0L)
  cat(r$report)
  # cross-device P2P (requires >= 2 GPUs)
  if (ggml_vulkan_device_count() >= 2) {
    r <- ggml_vulkan_p2p_selftest(0L, 1L)
    cat(r$report)
  }
}
```

`ggml_vulkan_shutdown` *Explicitly tear down the Vulkan instance and devices*

Description

Waits for all Vulkan devices to go idle, then destroys the devices and the Vulkan instance. Call this at the end of a script that used multiple GPUs (tensor / pipeline parallelism), *before* the process exits.

Usage

```
ggml_vulkan_shutdown(hard = FALSE, status = 0L)
```

Arguments

`hard`	Logical. If `TRUE`, call `_exit(status)` after teardown to skip all exit handlers and guarantee no exit-time segfault; never returns. Requires a build with `--enable-hard-exit` (see above); otherwise it warns and returns normally. Default `FALSE` (safe to call mid-session).
`status`	Integer process exit code used only when `hard = TRUE` (passed to `_exit()`). Defaults to `0L` (success). If you call this from an error path (e.g. a `tryCatch` handler after a stage failed), pass a non-zero `status` so the failed run does not exit `0` and mask the failure. Ignored when `hard = FALSE`.

Details

Why this exists: with several Vulkan devices active, R's own teardown at process exit can run *after* the Vulkan loader / Mesa ICD libraries have been unmapped, so the device destructors call into unmapped memory and the process crashes with a segfault *after* your results were already printed. Calling `ggml_vulkan_shutdown()` while the process is still alive runs the teardown while the loader is still mapped, avoiding that crash. It is only needed for exit cleanliness — your computed results are unaffected either way.

Idempotent and a no-op if Vulkan was never initialized. After this call the next Vulkan operation transparently re-initializes the instance from scratch, so it is safe to call mid-session too.

The plain teardown alone does not always win the exit-time race: with backends created inside pipeline / tensor-parallel forwards, the loader-static-destruction segfault (address ending `f1ba0`) can still fire flakily *after* your results are printed, because no R exit hook runs before R unmaps the loader. For a guaranteed-clean exit, pass `hard = TRUE` as the **last** statement of a standalone script: after teardown it calls `_exit(0)`, terminating the process immediately without running R's / the C runtime's exit handlers or unmapping any shared library — so there is no loader teardown phase left to crash. Because it bypasses R's normal shutdown, only use `hard = TRUE` at the very end of a script, never mid-session or in a package/interactive context.

Value

Invisibly `NULL`. Does not return when `hard = TRUE` *and* the hard-exit path is compiled in (see the section above).

Hard exit is opt-in at build time

The `_exit()` path is **compiled out by default**, because CRAN Repository Policy forbids a package from terminating the user's R session. In such a build `hard = TRUE` performs the normal teardown and emits a `warning` — it is never silently ignored — so the exit-time race described above can still fire. To compile it in, build from source with `R CMD INSTALL . --configure-args="--enable-hard-exit"` (on Windows set `Sys.setenv(GGML_VK_HARD_EXIT = "1")` first, since R ignores `configure.args` there). Check the current build with `ggml_vulkan_hard_exit_available()`.

See Also

`ggml_vulkan_hard_exit_available`

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 2) {
  W <- matrix(rnorm(2048 * 64), nrow = 2048)
  X <- matrix(rnorm(4 * 64), nrow = 4)
  Y <- ggml_vulkan_split_mul_mat(W, X, n_devices = 2)
  ggml_vulkan_shutdown()   # clean up before the script exits
}
```

---

ggml_vulkan_split_buffer_type

Create a Vulkan tensor-split buffer type

Description

Builds (or fetches from a cache) a ggml buffer type that row-splits a weight tensor across `n_devices` GPUs for tensor parallelism. This is the factory the graph allocator talks to (Stage E4): a weight placed in this buffer type has its rows distributed across the devices, each holding a contiguous band (see `ggml_vulkan_split_row_ranges`). Non-split tensors fall back to `main_device`.

Usage

```
ggml_vulkan_split_buffer_type(
  n_devices = ggml_vulkan_device_count(),
  main_device = 0L,
  weights = NULL,
  device_ids = NULL,
  transport = c("host-staging", "opaque-fd", "device-group"),
  probe = NULL
)
```

Arguments

n_devices	Number of GPUs to split across (default: all available). Ignored if device_ids is given (its length is used instead).
main_device	Device index (0-based) holding non-split fallbacks (default 0).
weights	Optional numeric vector of length n_devices giving the relative row share per device (e.g. c(3, 1)). NULL (default) splits evenly.
device_ids	Optional integer vector of physical GPU indices (0-based) the split slots map onto, e.g. c(2, 3). NULL (default) uses 0..n_devices-1. When supplied, n_devices is its length.
transport	Cross-device gather transport for the eventual compute: "host-staging" (default), "opaque-fd" or "device-group".
probe	A numeric vector c(N, K) describing a probe weight tensor (N rows, K columns) whose split allocation size to report, or NULL (default) to skip. Useful for inspecting how much total VRAM a given weight would occupy across the split.

Details

The buffer type is cached in C and lives for the rest of the session; the returned external pointer has no finalizer and must not be freed.

Value

A named list: ptr (external pointer to the cached buffer type), name (its config-derived name) and alloc_size (sum of the padded per-device slice sizes for probe, in bytes; 0 if no probe).

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 2) {
  bt <- ggml_vulkan_split_buffer_type(n_devices = 2, probe = c(2048, 64))
  bt$name
  bt$alloc_size   # total bytes for a 2048x64 f32 weight across 2 devices
}
```

---

ggml_vulkan_split_mul_mat

Tensor-parallel matrix multiply across Vulkan devices

Description

Computes Y = X %*% t(W) with the rows of the weight matrix W split across n_devices GPUs (true tensor parallelism), the activations X broadcast to every device, and the resulting column slices gathered back into a single matrix. This is Stage E3 of the Vulkan tensor-parallelism work: it exercises the row-split + per-device compute + cross-device gather path end to end on real hardware.

Usage

```
ggml_vulkan_split_mul_mat(
  W,
  X,
  n_devices = ggml_vulkan_device_count(),
  weights = NULL,
  device_ids = NULL,
  transport = c("host-staging", "opaque-fd", "device-group")
)
```

Arguments

W	Weight matrix, N x K (N output features, K input features). Its rows are split across the devices.
X	Activation matrix, M x K (M samples, K input features), broadcast to every device.
n_devices	Number of GPUs to split across (default: all available). Ignored if device_ids is given (its length is used instead).
weights	Optional numeric vector of length n_devices giving the relative share of W's rows per device (e.g. c(3, 1) for a 3:1 split). NULL (default) splits the rows evenly.
device_ids	Optional integer vector of physical GPU indices (0-based) to split across, e.g. c(2, 3) for the second replica of a TPxDP layout. NULL (default) uses devices 0..n_devices-1. When supplied, n_devices is taken from its length.
transport	Cross-device gather transport: "host-staging" (default, portable and correct on every driver), "opaque-fd" or "device-group" (see ggml_vulkan_p2p_selftest).

Details

Each device owns a contiguous band of W's rows (via the same row-split math as ggml_vulkan_split_row_ranges) and produces the matching band of Y's columns; the bands are gathered through the transport (host staging by default). This is a validation / correctness entry point, not a hot inference path — it stands up one backend per device per call and round-trips through host memory.

Value

The M x N result matrix, equal to X %*% t(W) up to the GPU's floating-point accumulation. Errors if the split compute fails; the C-level diagnostic is attached as attribute "report".

Examples

```
if (ggml_vulkan_available() && ggml_vulkan_device_count() >= 2) {
  W <- matrix(rnorm(8 * 4), nrow = 8)   # 8 output features x 4 inputs
  X <- matrix(rnorm(3 * 4), nrow = 3)   # 3 samples x 4 inputs
  Y <- ggml_vulkan_split_mul_mat(W, X, n_devices = 2)
  max(abs(Y - X %*% t(W)))              # ~1e-3 (GPU f16 accumulation)
}
```

`ggml_vulkan_split_row_ranges` *Compute tensor-parallel row-split ranges*

Description

Pure arithmetic helper for the Vulkan tensor-parallel split buffer type: given a number of tensor rows and a per-device weight vector, returns the half-open row range `[row_low, row_high)` owned by each device. Row boundaries are rounded down to a fixed granularity, and the last device always covers up to `nrows`, so the ranges are contiguous, non-overlapping and cover every row exactly once.

Usage

```
ggml_vulkan_split_row_ranges(nrows, n_devices, weights = NULL)
```

Arguments

`nrows`	Number of rows in the tensor (integer).
`n_devices`	Number of devices to split across (integer, >= 1).
`weights`	Optional numeric vector of length `n_devices` giving the relative share of rows per device. `NULL` (default) splits evenly.

Details

This touches no GPU and is exposed mainly to verify the split logic. It is the math behind row-split tensor parallelism (a weight matrix distributed across several GPUs), independent of any actual multi-GPU allocation.

Value

A named list with `row_low` and `row_high`: numeric vectors of length `n_devices` holding 0-based, half-open row ranges.

Examples

```
if (ggml_vulkan_available()) {
  # 4096 rows across 2 devices, evenly
  ggml_vulkan_split_row_ranges(4096L, 2L)
  # weighted 3:1
  ggml_vulkan_split_row_ranges(4096L, 2L, weights = c(3, 1))
}
```

ggml_vulkan_status *Print Vulkan status*

Description

Prints information about Vulkan availability and devices, and returns the same information as a list so it can be used programmatically.

Usage

```
ggml_vulkan_status()
```

Value

Invisibly, a list with components:

available Logical, whether Vulkan support is compiled in and usable

n_devices Integer, number of Vulkan devices (0 if unavailable)

devices List of per-device property lists as returned by ggml_vulkan_list_devices; an empty list if there are none

Status is also printed to the console.

Examples

```
ggml_vulkan_status()

st <- ggml_vulkan_status()
if (st$available && st$n_devices > 0) {
  print(st$devices[[1]]$name)
}
```

ggml_win_part *Window Partition (Graph)*

Description

Partitions a tensor into non-overlapping windows of size w.

Usage

```
ggml_win_part(ctx, a, w)
```

Arguments

ctx	GGML context
a	Input tensor
w	Window size

Value

Partitioned tensor

---

ggml_win_unpart *Window Un-partition (Graph)*

Description

Reassembles windowed partitions produced by ggml_win_part.

Usage

```
ggml_win_unpart(ctx, a, w0, h0, w)
```

Arguments

ctx	GGML context
a	Input tensor
w0	Original width
h0	Original height
w	Window size

Value

Un-partitioned tensor

ggml_with_temp_ctx *Execute with Temporary Context*

Description

Creates a temporary context, executes code, and frees it automatically. Useful when you need to create large temporary tensors.

Usage

```
ggml_with_temp_ctx(mem_size, expr)
```

Arguments

mem_size	Context memory size in bytes
expr	Expression to evaluate with the temporary context

Value

Result of the expression

Examples

```
# Create tensors in temporary context
result <- ggml_with_temp_ctx(1024 * 1024, {
  a <- ggml_new_tensor_1d(ctx, GGML_TYPE_F32, 10)
  ggml_set_f32(a, 1:10)
  ggml_get_f32(a)
})
```

gguf_free *Free GGUF Resources*

Description

Explicitly frees the internal GGUF context. Called automatically by the garbage collector, but can be called manually to release memory sooner.

Usage

```
gguf_free(x)
```

Arguments

x	A gguf object.

Value

Called for its side effect (releases the GGUF context); invisibly returns NULL.

---

gguf_load *Load a GGUF File*

Description

Opens a GGUF file and reads its metadata. By default also reads tensor data into memory; with meta_only = TRUE only the header and key-value metadata are read (no tensor data is allocated), which is cheap and enough for inspecting architecture / type fields. Returns an S3 object of class "gguf" wrapping the internal pointer.

Usage

```
gguf_load(path, meta_only = FALSE)
```

Arguments

path	Path to a .gguf file.
meta_only	If TRUE, read only the header and metadata without allocating tensor data. Metadata, tensor names and tensor info remain available; gguf_tensor_data is not (reload with meta_only = FALSE). Default FALSE.

Value

An object of class "gguf".

---

gguf_metadata *Get GGUF Metadata*

Description

Returns all key-value metadata pairs from a GGUF file as a named list.

Usage

```
gguf_metadata(x)
```

Arguments

x	A gguf object from gguf_load.

Value

A named list of metadata values.

gguf_tensor_data *Extract Tensor Data*

Description

Dequantizes (if needed) and returns tensor weights as an R numeric array with dimensions matching the tensor shape.

Usage

```
gguf_tensor_data(x, name)
```

Arguments

x	A gguf object.
name	Tensor name (character).

Value

A numeric array.

gguf_tensor_info *Get Tensor Info*

Description

Returns name, shape, type, and size in bytes for a single tensor.

Usage

```
gguf_tensor_info(x, name)
```

Arguments

x	A gguf object.
name	Tensor name (character).

Details

When the file was opened with meta_only = TRUE, the per-dimension shape is NA (the public GGUF API does not expose tensor dimensions without allocating tensors); name, type and size_bytes are still returned.

Value

A list with elements name, shape, type, size_bytes.

gguf_tensor_names *List Tensor Names in a GGUF File*

Description

List Tensor Names in a GGUF File

Usage

```
gguf_tensor_names(x)
```

Arguments

x	A gguf object.

Value

Character vector of tensor names.

glance.ggmlr_parsnip_model *One-row summary of a fitted ggml parsnip model*

Description

Returns a single-row [tibble][tibble::tibble] summarising the fitted model, in broom ‘glance()‘ style.

Usage

```
## S3 method for class 'ggmlr_parsnip_model'
glance(x, ...)
```

Arguments

x	A fitted ‘ggmlr_parsnip_model‘ (the engine object inside a parsnip fit; e.g. from ‘extract_fit_engine()‘).
...	Unused; for generic compatibility.

Value

A one-row tibble with columns: ‘mode‘, ‘n_features‘, ‘n_layers‘, ‘total_params‘, ‘optimizer‘, ‘loss‘, ‘backend‘, ‘epochs‘, ‘fit_time‘ (wall seconds) and ‘final_loss‘ (last training loss, ‘NA‘ if no history).

Examples

```
ggml_set_n_threads(1L)  # deterministic, single OpenMP pool
spec <- parsnip::mlp(hidden_units = 8L, epochs = 3L) |>
  parsnip::set_engine("ggml", backend = "cpu") |>
  parsnip::set_mode("regression")
fit_obj <- parsnip::fit(spec, mpg ~ ., data = mtcars)
generics::glance(parsnip::extract_fit_engine(fit_obj))
```

---

iq2xs_free_impl *Free IQ2 Quantization Tables*

Description

Frees lookup tables for IQ2 quantization types.

Usage

```
iq2xs_free_impl(type)
```

Arguments

type	GGML type constant

Value

NULL invisibly

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

iq2xs_init_impl *Initialize IQ2 Quantization Tables*

Description

Initializes lookup tables for IQ2 quantization types. Must be called before using iq2_xxs, iq2_xs, or iq2_s quantization.

Usage

```
iq2xs_init_impl(type)
```

Arguments

type	GGML type constant (e.g., GGML_TYPE_IQ2_XXS())

Value

NULL invisibly

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

iq3xs_free_impl *Free IQ3 Quantization Tables*

Description

Frees lookup tables for IQ3 quantization types.

Usage

```
iq3xs_free_impl(grid_size)
```

Arguments

grid_size	Grid size for IQ3

Value

NULL invisibly

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

iq3xs_init_impl *Initialize IQ3 Quantization Tables*

Description

Initializes lookup tables for IQ3 quantization types. Must be called before using iq3_xxs or iq3_s quantization.

Usage

```
iq3xs_init_impl(grid_size)
```

Arguments

grid_size	Grid size for IQ3 (typically 256)

Value

NULL invisibly

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

`lr_scheduler_cosine` *Cosine-annealing learning rate scheduler*

Description

Varies the learning rate following a cosine curve from `lr_max` down to `lr_min` over `T_max` steps. Restarts (SGDR-style) if `restart = TRUE`.

Usage

```
lr_scheduler_cosine(optimizer, T_max, lr_min = 0, restart = FALSE, T_mult = 1)
```

Arguments

`optimizer`	Optimizer environment.
`T_max`	Number of steps for the first cosine cycle.
`lr_min`	Minimum learning rate (default 0).
`restart`	Logical; if `TRUE` restart after each cycle.
`T_mult`	Cycle-length multiplier applied after every restart (default 1, i.e. constant period). Only meaningful when `restart = TRUE`.

Details

With `restart = TRUE` and `T_mult > 1` this is the full *CosineAnnealingWarmRestarts* (SGDR) schedule: the first cycle lasts `T_max` steps and every subsequent cycle is `T_mult` times longer than the previous one. `T_mult = 1` (the default) keeps every cycle the same length.

Value

An `lr_scheduler_cosine` environment

References

Loshchilov, I. & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. ICLR.

Examples

```
w   <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_adam(list(w = w), lr = 0.1)
sch <- lr_scheduler_cosine(opt, T_max = 50L)
for (epoch in 1:50) sch$step()

# SGDR with doubling cycle length: 10, 20, 40, ...
opt2 <- optimizer_adam(list(w = w), lr = 0.1)
sch2 <- lr_scheduler_cosine(opt2, T_max = 10L, restart = TRUE, T_mult = 2)
for (epoch in 1:70) sch2$step()
```

`lr_scheduler_cyclic` *Cyclical learning rate scheduler*

Description

Cycles the learning rate linearly between `base_lr` and `max_lr` over a triangular cycle of `2 * step_size_up` steps (Smith, 2017). Each call to `$step()` advances one step — call it **per batch**, not per epoch.

Usage

```
lr_scheduler_cyclic(
  optimizer,
  base_lr,
  max_lr,
  step_size_up = 2000L,
  step_size_down = NULL,
  mode = c("triangular", "triangular2", "exp_range"),
  gamma = 1
)
```

Arguments

`optimizer`	Optimizer environment.
`base_lr`	Lower learning rate bound.
`max_lr`	Upper learning rate bound.
`step_size_up`	Steps in the increasing half of a cycle (default 2000).
`step_size_down`	Steps in the decreasing half. `NULL` (default) mirrors `step_size_up`.
`mode`	One of `"triangular"` (default), `"triangular2"` (amplitude halved each cycle) or `"exp_range"` (amplitude scaled by `gamma^step`).
`gamma`	Decay factor for `mode = "exp_range"` (default 1.0).

Value

An `lr_scheduler_cyclic` environment with `$step()` and `$get_lr()`.

References

Smith, L. N. (2017). Cyclical Learning Rates for Training Neural Networks. WACV.

Examples

```
w   <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_sgd(list(w = w), lr = 0.01)
sch <- lr_scheduler_cyclic(opt, base_lr = 0.001, max_lr = 0.01,
                           step_size_up = 10L)
for (i in 1:40) sch$step()
```

`lr_scheduler_onecycle` *One-cycle learning rate scheduler*

Description

Implements the 1cycle policy (Smith, 2018): the learning rate rises from `max_lr / div_factor` to `max_lr` during the first `pct_start` fraction of training, then anneals down to `max_lr / final_div_factor`. Call `$step()` **per batch**; `total_steps` is the total number of batches over the whole run (`epochs * batches_per_epoch`).

Usage

```
lr_scheduler_onecycle(
  optimizer,
  max_lr,
  total_steps,
  pct_start = 0.3,
  div_factor = 25,
  final_div_factor = 10000,
  anneal_strategy = c("cos", "linear"),
  cycle_momentum = TRUE,
  base_momentum = 0.85,
  max_momentum = 0.95
)
```

Arguments

`optimizer`	Optimizer environment.
`max_lr`	Peak learning rate.
`total_steps`	Total number of `$step()` calls for the whole run.
`pct_start`	Fraction of `total_steps` spent increasing the LR (default 0.3).
`div_factor`	Initial LR is `max_lr / div_factor` (default 25).
`final_div_factor`	Final LR is `max_lr / final_div_factor` (default 1e4).
`anneal_strategy`	`"cos"` (default) or `"linear"`.
`cycle_momentum`	Logical; cycle momentum inversely to LR (default `TRUE`).
`base_momentum`, `max_momentum`	Momentum bounds used when `cycle_momentum = TRUE` (defaults 0.85 / 0.95).

Details

When `cycle_momentum = TRUE` the momentum-like hyper-parameter moves inversely to the learning rate: `$momentum` for SGD, `$beta1` for Adam. For other optimizers momentum cycling is skipped.

Value

An lr_scheduler_onecycle environment with $step() and $get_lr().

References

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters. arXiv:1803.09820.

Examples

```
w   <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_sgd(list(w = w), lr = 0.01, momentum = 0.9)
sch <- lr_scheduler_onecycle(opt, max_lr = 0.1, total_steps = 100L)
for (i in 1:100) sch$step()
```

---

lr_scheduler_step *Step-decay learning rate scheduler*

Description

Multiplies the optimizer learning rate by gamma every step_size calls to $step().

Usage

```
lr_scheduler_step(optimizer, step_size, gamma = 0.1)
```

Arguments

optimizer	An ag_optimizer_adam or ag_optimizer_sgd environment.
step_size	Decay every this many steps (epochs).
gamma	Multiplicative decay factor (default 0.1).

Value

An lr_scheduler_step environment

Examples

```
w   <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_adam(list(w = w), lr = 0.1)
sch <- lr_scheduler_step(opt, step_size = 10L, gamma = 0.5)
for (epoch in 1:30) sch$step()
opt$lr  # 0.1 * 0.5^3 = 0.0125
```

lr_scheduler_warmup_cosine *Linear warmup followed by cosine annealing*

Description

Raises the learning rate linearly from `warmup_start_lr` to the optimizer's initial `lr` over `warmup_steps` steps, then anneals it to `lr_min` following a cosine curve over the remaining `total_steps - warmup_steps` steps. This is the standard transformer training schedule. Call `$step()` **per batch**.

Usage

```
lr_scheduler_warmup_cosine(
  optimizer,
  warmup_steps,
  total_steps,
  lr_min = 0,
  warmup_start_lr = 0
)
```

Arguments

`optimizer`	Optimizer environment. Its current `lr` is taken as the peak learning rate.
`warmup_steps`	Number of linear warmup steps.
`total_steps`	Total number of `$step()` calls (warmup included).
`lr_min`	Final learning rate after annealing (default 0).
`warmup_start_lr`	Learning rate at step 0 (default 0).

Value

An `lr_scheduler_warmup_cosine` environment with `$step()` and `$get_lr()`.

Examples

```
w   <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_adam(list(w = w), lr = 1e-3)
sch <- lr_scheduler_warmup_cosine(opt, warmup_steps = 10L, total_steps = 100L)
for (i in 1:100) sch$step()
```

nn_topo_sort *Topologically sort nodes reachable from output nodes*

Description

Topologically sort nodes reachable from output nodes

Usage

```
nn_topo_sort(outputs)
```

Arguments

outputs	List of output ggml_tensor_node objects

Value

Named list: nodes in topological order (inputs first, outputs last)

onnx_device_info *ONNX model device/scheduler diagnostics*

Description

Returns information about backend placement: which backends are available, how the scheduler splits the graph, and how many ops are supported by GPU vs CPU-only.

Usage

```
onnx_device_info(model)
```

Arguments

model	An `onnx_model` object from `onnx_load()`.

Value

A list with:

backends Character vector of backend names (e.g. `"Vulkan0"`, `"CPU"`)

n_backends Number of backends

n_splits Number of scheduler splits (1 = all on one backend)

n_nodes Total graph nodes

gpu_ops Ops supported by GPU backend

cpu_ops Ops that can only run on CPU

cpu_only_ops Named integer vector: op type => count (empty if all on GPU)

onnx_graph_info *ONNX graph structure*

Description

The parsed graph as a table: one row per node, with its operator type and the names of its inputs and outputs. Use it to find which node produces a given edge, or what an edge feeds into, without re-parsing the file.

Usage

```
onnx_graph_info(model, producer_of = NULL, consumer_of = NULL)
```

Arguments

model	An onnx_model object from onnx_load().
producer_of	Optional edge name. When given, returns the rows whose node lists that name among its outputs.
consumer_of	Optional edge name. When given, returns the rows whose node lists that name among its inputs.

Details

Reading the structure beats searching the .onnx binary for a name: a name occurs both as a producer's output and as its consumers' input, and telling those apart by hand means getting the protobuf field numbers right every time.

Value

A data frame with columns index, op_type, output (the first output) and list-columns outputs and inputs.

Examples

```
if (file.exists("model.onnx")) {
  m <- onnx_load("model.onnx")
  g <- onnx_graph_info(m)
  head(g)
  onnx_graph_info(m, producer_of = "881")
}
```

onnx_inputs *List ONNX model inputs*

Description

Returns the names and shapes of model inputs (excluding weight initializers). Use this to know what to pass to `onnx_run()`.

Usage

```
onnx_inputs(model)
```

Arguments

`model`	An `onnx_model` object from `onnx_load()`.

Value

A named list where names are input tensor names and values are integer vectors of dimension sizes (-1 for dynamic dimensions).

onnx_load *Load an ONNX model*

Description

Parses an .onnx file, builds a ggml computation graph, and allocates tensors on the specified device. Weights are loaded via memory-mapped file (zero-copy where possible).

Usage

```
onnx_load(
  path,
  device = NULL,
  input_shapes = NULL,
  n_threads = NULL,
  dtype = "f32"
)
```

Arguments

path	Path to .onnx file.
device	Backend device: "vulkan" (default if available) or "cpu".
input_shapes	Optional named list of integer vectors specifying fixed shapes for inputs with dynamic dimensions. Names must match input tensor names. Each shape must include all dimensions including batch, e.g. list(image = c(1L, 3L, 224L, 224L)). Required when the model has dynamic dimensions and no default shape.
n_threads	Number of CPU threads. NULL (default) reads getOption("ggmlR.n_threads"); if that is also unset, uses parallel::detectCores() - 1 (minimum 1).
dtype	Weight precision: "f32" (default) or "f16". When "f16", large weight tensors (>= 256 elements) are stored in half-precision for faster Vulkan compute and lower VRAM usage. Small tensors (bias, scalars, batch-norm params) remain in F32 for numerical stability. Inputs and outputs are always F32.

Value

An opaque model object (external pointer) for use with onnx_run(), onnx_summary(), and onnx_inputs().

---

onnx_run *Run ONNX model inference*

Description

Run ONNX model inference

Usage

```
onnx_run(model, inputs)
```

Arguments

model	An onnx_model object from onnx_load().
inputs	A named list of numeric vectors/matrices. Names must match the model's input tensor names. Use onnx_inputs() to see expected names and shapes.

Value

A named list of output tensors: numeric vectors carrying a dim attribute when the output has more than one dimension.

The buffers are row-major, as ONNX stores them, so dim is given in storage order – fastest-varying axis first – which is the reverse of the shape the ONNX file declares. An output declared [nbox, 4] arrives as dim = c(4, nbox), one detection per column. This keeps dim an honest description of the memory, so ordinary R indexing is correct; use predict.onnx_model to get arrays in the declared ONNX order instead.

onnx_summary *ONNX model summary*

Description

Returns metadata about a loaded ONNX model.

Usage

```
onnx_summary(model)
```

Arguments

model	An onnx_model object from onnx_load().

Value

A list with ir_version, opset_version, producer, graph_name, n_nodes, n_initializers, and ops.

optimizer_adam *Create an Adam optimizer*

Description

Create an Adam optimizer

Usage

```
optimizer_adam(
  params,
  lr = 0.001,
  beta1 = 0.9,
  beta2 = 0.999,
  eps = 1e-08,
  accumulate_steps = 1L,
  average = TRUE
)
```

Arguments

params	List of ag_param tensors, named or not (names are used only for display). The same tensor listed twice is kept once, with a warning.
lr	Learning rate (default 1e-3)
beta1	First moment decay (default 0.9)
beta2	Second moment decay (default 0.999)
eps	Stability constant (default 1e-8)
accumulate_steps	Number of backward() passes to accumulate before a call to $step() updates the weights (default 1, no accumulation). The bias-correction counter advances only on a real update, so accumulating k micro-batches behaves like one step on the larger batch rather than like k steps.
average	When accumulating, divide the accumulated gradient by accumulate_steps (default TRUE). See optimizer_sgd.

Value

An optimizer environment with step(grads), zero_grad(), state() and load_state(state). state() returns a versioned list (moments as host double matrices, t, accum_count); load_state() checks the type, parameter count, names (when both sides have them) and shapes before writing anything, restores t and accum_count, and does NOT restore hyperparameters (it warns when they differ). Exact continuation holds on the same device and dtype; CPU <-> GPU in f32 continues within float tolerance; across f16 and f32 it cannot be exact. Only read saved states you trust (readRDS).

Examples

```
w <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_adam(list(w = w), lr = 1e-3)
```

---

optimizer_sgd *Create an SGD optimizer*

Description

Create an SGD optimizer

Usage

```
optimizer_sgd(
  params,
  lr = 0.01,
  momentum = 0,
  accumulate_steps = 1L,
  average = TRUE
)
```

Arguments

params	List of ag_param tensors, named or not (names are used only for display). The same tensor listed twice is kept once, with a warning.
lr	Learning rate (default 0.01)
momentum	Momentum factor (default 0)
accumulate_steps	Number of backward() passes to accumulate before a call to $step() updates the weights (default 1, no accumulation). With k > 1 the first k - 1 calls to $step() only count, and the k-th applies the update and resets the counter, so a micro-batch loop needs no bookkeeping of its own.
average	When accumulating, divide the accumulated gradient by accumulate_steps (default TRUE). That is what makes k micro-batches equivalent to one batch of k times the size for an averaging loss such as ag_mse_loss; set it to FALSE for a summing loss, where the plain sum is already right.

Value

An optimizer environment with step(grads), zero_grad(), state() and load_state(state). state() returns a versioned list (moments as host double matrices, t, accum_count); load_state() checks the type, parameter count, names (when both sides have them) and shapes before writing anything, restores t and accum_count, and does NOT restore hyperparameters (it warns when they differ). Exact continuation holds on the same device and dtype; CPU <-> GPU in f32 continues within float tolerance; across f16 and f32 it cannot be exact. Only read saved states you trust (readRDS).

Examples

```
w <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_sgd(list(w = w), lr = 0.01)

# accumulate 4 micro-batches per update
opt4 <- optimizer_sgd(list(w = w), lr = 0.01, accumulate_steps = 4L)
```

---

plot.ggml_history *Plot training history*

Description

Plots loss and accuracy curves over epochs.

Usage

```
## S3 method for class 'ggml_history'
plot(x, ...)
```

Arguments

x	A ggml_history object
...	Additional arguments (ignored)

Value

The history object (invisibly).

---

predict.ggml_sequential_model
Predict with a Trained Model

Description

Generates predictions from a trained model. Uses the standard R `predict` generic for compatibility with keras3 and the broader R ecosystem.

Usage

```
## S3 method for class 'ggml_sequential_model'
predict(object, x, batch_size = NULL, ...)

## S3 method for class 'ggml_functional_model'
predict(object, x, batch_size = NULL, ...)
```

Arguments

object	A trained model object.
x	Input data (matrix, array, or list for multi-input models).
batch_size	Batch size for inference. NULL (default) uses 32, as in keras3.
...	Additional arguments (ignored).

Value

Matrix of predictions.

`predict.onnx_model` *Predict with an ONNX Model*

Description

Runs inference over `x`, batching it to the fixed input shape the model was loaded with. This is the keras-compatible entry point: together with the `ggml_sequential_model` and `ggml_functional_model` methods it gives ggmlR and ONNX models a single `predict()` interface.

Usage

```
## S3 method for class 'onnx_model'
predict(object, x, batch_size = NULL, ...)
```

Arguments

`object`	An `onnx_model` from `onnx_load`.
`x`	Input data: an array or matrix whose first dimension indexes samples, or a named list of such arrays for multi-input models.
`batch_size`	Ignored, with a warning if not `NULL`. The batch size is fixed when the model is loaded; pass `input_shapes` to `onnx_load` to change it. The argument exists so the signature matches the other `predict()` methods.
`...`	Ignored.

Details

Unlike `onnx_run`, which takes a named list of inputs and always returns a named list of outputs, this method takes an array (or a named list for multi-input models) and unwraps a single output. The output keeps the dimensions the model produced: a 2D output comes back as a matrix, a 4D output stays 4D.

An ONNX graph is built for one fixed batch size at `onnx_load` time and cannot be resized afterwards, so `x` is split into chunks of that size. A trailing partial chunk is padded to a full batch and the padding is dropped from the result.

Value

For a single-output model, an array with the model's output dimensions and `nrow(x)` samples in the first one (a matrix when the output is 2D). For a multi-output model, a named list of such arrays.

See Also

`onnx_run` for the lower-level named-list interface.

print.ag_tensor *Print method for ag_tensor*

Description

Print method for ag_tensor

Usage

```
## S3 method for class 'ag_tensor'
print(x, ...)
```

Arguments

x	An ag_tensor
...	Ignored

Value

The input x, returned invisibly (called for its side effect of printing).

print.ggml_functional_model *Print method for ggml_functional_model*

Description

Print method for ggml_functional_model

Usage

```
## S3 method for class 'ggml_functional_model'
print(x, ...)
```

Arguments

x	A ggml_functional_model object
...	Additional arguments (ignored)

Value

The model object (invisibly).

print.ggml_history *Print method for ggml_history*

Description

Print method for ggml_history

Usage

```
## S3 method for class 'ggml_history'
print(x, ...)
```

Arguments

x	A ggml_history object
...	Additional arguments (ignored)

Value

The history object (invisibly).

print.ggml_sequential_model *Print method for ggml_sequential_model*

Description

Prints a summary of the model architecture including layer types, output shapes, and parameter counts.

Usage

```
## S3 method for class 'ggml_sequential_model'
print(x, ...)
```

Arguments

x	A ggml_sequential_model object
...	Additional arguments (ignored)

Value

The model object (invisibly).

print.onnx_model *Print ONNX model summary*

Description

Print ONNX model summary

Usage

```
## S3 method for class 'onnx_model'
print(x, ...)
```

Arguments

x	An onnx_model object.
...	Ignored.

Value

Invisibly returns x.

pt_cube4_layout *The piece layout of a 4x4x4 cube*

Description

Which stickers make up each piece, for a cube written as 96 stickers in face order. Pieces come in three kinds and the model is told which is which: 8 corners of three stickers, 24 wings of two, 24 centres of one. The short pieces are padded to three slots and the padding masked off.

Usage

```
pt_cube4_layout()
```

Details

Sticker numbers are 0-based, as the checkpoint's own layout table has them.

Value

A list with positions (a 56 x 3 integer matrix of sticker indices), mask (56 x 3 logical, which slots are real), types (length-56 integer, 0 corner / 1 wing / 2 centre), state_size and num_classes.

See Also

pt_transformer

Examples

```
lay <- pt_cube4_layout()
dim(lay$positions)
table(lay$types)
```

`pt_forward` *Run a piece transformer*

Description

Reads a state, or a batch of them, and returns a value per action.

Usage

```
pt_forward(model, state)
```

Arguments

`model`	A `pt_transformer`, normally from `pt_transformer_load`.
`state`	Colours in `0..num_classes-1`: an integer vector of `state_size` for one state, or a matrix with `state_size` columns — one state per row — for a batch. A solved 4x4x4 is `rep(0:5, each = 16)`.

Value

For one state, a numeric vector of `n_actions` values, one per move. For a batch, a matrix with one row per state and one column per action.

One state or many

A batch is not a convenience here, it is the way to get the work done quickly. Every layer but the attention treats a row on its own, so a batch of states is the same arithmetic on taller matrices — and a tall matmul is what BLAS is good at, where a short one spends its time in the call rather than the multiply. Scoring the ten states of a beam in one call is several times faster than scoring them one after another.

See Also

`pt_transformer_load`

Examples

```
## Not run:
model <- pt_transformer_load("model.pth")
solved <- rep(0:5, each = 16)

pt_forward(model, solved)                       # 24 values
pt_forward(model, rbind(solved, solved))        # 2 x 24
```

```
## End(Not run)
```

`pt_transformer` *Build a piece transformer*

Description

Creates the model with its weights at their initial values. To run a trained one, use `pt_transformer_load`, which builds this and then fills it from a checkpoint.

Usage

```
pt_transformer(
  layout,
  d_model = 256L,
  n_heads = 8L,
  n_layers = 4L,
  ff_dim = 1024L,
  n_actions = 24L,
  activation = "relu"
)
```

Arguments

`layout`	A layout list, as `pt_cube4_layout` returns.
`d_model`	Width of the model (default 256).
`n_heads`	Attention heads (default 8).
`n_layers`	Encoder blocks (default 4).
`ff_dim`	Width of the feed-forward hidden layer (default 1024).
`n_actions`	Size of the output — one value per action (default 24).
`activation`	Feed-forward activation, `"relu"` (default) or `"gelu"` / `"silu"`.

Value

A `pt_transformer` object.

See Also

`pt_transformer_load`, `pt_forward`

pt_transformer_load *Build a piece transformer and load a checkpoint into it*

Description

Reads a `.pth` saved by `torch.save()` and returns the model ready to run. Everything that differs between torch's conventions and this package's is resolved here rather than at inference time.

Usage

```
pt_transformer_load(path, layout = pt_cube4_layout(), ...)
```

Arguments

`path`	Path to the `.pth` checkpoint.
`layout`	A layout list; defaults to `pt_cube4_layout`.
`...`	Passed to `pt_transformer` — `d_model`, `n_heads` and the rest, when they differ from the defaults.

Value

A `pt_transformer` with the checkpoint's weights.

See Also

`pth_catalogue` to see what a checkpoint holds.

Examples

```
## Not run:
model <- pt_transformer_load("model.pth")
q <- pt_forward(model, rep(0:5, each = 16))
which.max(q)

## End(Not run)
```

pth_catalogue *List the tensors in a PyTorch checkpoint*

Description

Reads the metadata of a `.pth` file — the tensor names, shapes and dtypes — without reading the tensors. A checkpoint is a zip holding one pickle of metadata and one entry per tensor, so the catalogue is cheap whatever the file weighs.

Usage

```
pth_catalogue(path)
```

Arguments

path	Path to a .pth file saved by torch.save().

Details

What comes back is the checkpoint's own state_dict order, which is the order the model's modules were built in.

Value

A data.frame with one row per tensor and the columns name, shape (as "a x b"), dtype, n (element count), bytes, and entry (the zip entry holding the data). The attribute "file" carries the path.

Examples

```
## Not run:
cat <- pth_catalogue("model.pth")
head(cat)
sum(cat$n)          # total parameters

## End(Not run)
```

---

pth_load *Read the tensors of a PyTorch checkpoint*

Description

Reads a .pth file saved by torch.save() into plain R arrays. No Python and no torch: a checkpoint is a zip of one metadata pickle and one entry per tensor, and both are read here directly.

Usage

```
pth_load(path, names = NULL)
```

Arguments

path	Path to a .pth file saved by torch.save().
names	Optional character vector of tensor names to read. The default reads all of them. Use pth_catalogue to see what is there.

Value

A named list of numeric arrays, in the checkpoint's own order. Scalars come back as length-1 vectors; everything else carries a dim. The attribute "file" carries the path.

Row-major, and what that means for shapes

PyTorch stores a tensor row-major and R stores an array column-major, so the same bytes describe transposed arrays in the two languages. The tensors come back with their dimensions *reversed* — a PyTorch `[768, 256]` weight arrives as a `256 x 768` R matrix — which keeps the elements where they belong without moving a single one. For a linear layer that is the happy case: `x %*% w` is then the layer's own arithmetic, no transpose needed.

See Also

`pth_catalogue` to list a checkpoint without reading it.

Examples

```
## Not run:
w <- pth_load("model.pth")
dim(w$output_layer.weight)

# just the one tensor
pth_load("model.pth", names = "output_layer.bias")

## End(Not run)
```

---

quantize_iq2_xxs *Quantize Data (IQ)*

Description

Quantizes float data to IQ format. IQ formats require importance matrix initialization before use (see iq2xs_init_impl, iq3xs_init_impl).

Usage

```
quantize_iq2_xxs(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq2_xs(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq2_s(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq3_xxs(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq3_s(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq1_s(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq1_m(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq4_nl(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_iq4_xs(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row
imatrix	Optional importance matrix (numeric vector or NULL)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_mxfp4 *Quantize Data (MXFP4)*

Description

Quantizes float data to MXFP4 (microscaling FP4) format.

Usage

```
quantize_mxfp4(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row
imatrix	Optional importance matrix (numeric vector or NULL)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_nvfp4 *Quantize Data (NVFP4)*

Description

Quantizes float data to NVFP4 format (NVIDIA FP4 with UE4M3 per-sub-block scale).

Usage

```
quantize_nvfp4(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row (must be multiple of 64)
imatrix	Optional importance matrix (currently ignored)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

quantize_q1_0 *Quantize Data (Q1_0)*

Description

Quantizes float data to Q1_0 format (1-bit-per-weight sign quantization).

Usage

```
quantize_q1_0(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row (must be multiple of 128)
imatrix	Optional importance matrix (currently ignored)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

quantize_q2_K *Quantize Data (K-quants)*

Description

Quantizes float data to K-quant format with optional importance matrix. K-quants provide better quality/size tradeoffs than basic quants.

Usage

```
quantize_q2_K(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q3_K(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q4_K(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q5_K(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q6_K(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row
imatrix	Optional importance matrix (numeric vector or NULL)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_q4_0 *Quantize Data (Q4_0)*

Description

Quantizes float data to Q4_0 format with optional importance matrix.

Usage

```
quantize_q4_0(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q4_1(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q5_0(src_data, n_rows, n_per_row, imatrix = NULL)
```

```
quantize_q5_1(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_q8_0(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row
imatrix	Optional importance matrix (numeric vector or NULL)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_row_iq3_xxs_ref

Quantize Row Reference (IQ)

Description

Basic row-level IQ quantization.

Usage

```
quantize_row_iq3_xxs_ref(src_data, n_elements)

quantize_row_iq4_nl_ref(src_data, n_elements)

quantize_row_iq4_xs_ref(src_data, n_elements)

quantize_row_iq3_s_ref(src_data, n_elements)

quantize_row_iq2_s_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_row_mxfp4_ref

Quantize Row Reference (MXFP4)

Description

Basic row-level MXFP4 quantization.

Usage

```
quantize_row_mxfp4_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

quantize_row_nvfp4_ref *Quantize Row to NVFP4 (Reference)*

Description

Basic row-level NVFP4 quantization.

Usage

```
quantize_row_nvfp4_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

quantize_row_q1_0_ref *Quantize Row to Q1_0 (Reference)*

Description

Basic row-level Q1_0 quantization.

Usage

```
quantize_row_q1_0_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_row_q2_K_ref *Quantize Row Reference (K-quants)*

Description

Basic row-level K-quant quantization without importance matrix.

Usage

```
quantize_row_q2_K_ref(src_data, n_elements)

quantize_row_q3_K_ref(src_data, n_elements)

quantize_row_q4_K_ref(src_data, n_elements)

quantize_row_q5_K_ref(src_data, n_elements)

quantize_row_q6_K_ref(src_data, n_elements)

quantize_row_q8_K_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

---

quantize_row_q4_0_ref *Quantize Row Reference (Basic)*

Description

Basic row-level quantization without importance matrix. These are reference implementations.

Usage

```
quantize_row_q4_0_ref(src_data, n_elements)

quantize_row_q4_1_ref(src_data, n_elements)

quantize_row_q5_0_ref(src_data, n_elements)

quantize_row_q5_1_ref(src_data, n_elements)

quantize_row_q8_0_ref(src_data, n_elements)

quantize_row_q8_1_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_tq1_0_ref(), quantize_tq1_0()

quantize_row_tq1_0_ref *Quantize Row Reference (Ternary)*

Description

Basic row-level ternary quantization.

Usage

```
quantize_row_tq1_0_ref(src_data, n_elements)

quantize_row_tq2_0_ref(src_data, n_elements)
```

Arguments

src_data	Numeric vector of float values to quantize
n_elements	Number of elements to quantize

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_tq1_0()

quantize_tq1_0 *Quantize Data (Ternary)*

Description

Quantizes float data to ternary format with optional importance matrix.

Usage

```
quantize_tq1_0(src_data, n_rows, n_per_row, imatrix = NULL)

quantize_tq2_0(src_data, n_rows, n_per_row, imatrix = NULL)
```

Arguments

src_data	Numeric vector of float values to quantize
n_rows	Number of rows
n_per_row	Number of elements per row
imatrix	Optional importance matrix (numeric vector or NULL)

Value

Raw vector of quantized data

See Also

Other quantization: dequantize_row_iq2_xxs(), dequantize_row_mxfp4(), dequantize_row_nvfp4(), dequantize_row_q1_0(), dequantize_row_q2_K(), dequantize_row_q4_0(), dequantize_row_tq1_0(), ggml_quant_block_info(), iq2xs_free_impl(), iq2xs_init_impl(), iq3xs_free_impl(), iq3xs_init_impl(), quantize_iq2_xxs(), quantize_mxfp4(), quantize_nvfp4(), quantize_q1_0(), quantize_q2_K(), quantize_q4_0(), quantize_row_iq3_xxs_ref(), quantize_row_mxfp4_ref(), quantize_row_nvfp4_ref(), quantize_row_q1_0_ref(), quantize_row_q2_K_ref(), quantize_row_q4_0_ref(), quantize_row_tq1_0_ref()

---

rope_types *RoPE Mode Constants*

Description

RoPE (Rotary Position Embedding) Type Constants

Usage

```
GGML_ROPE_TYPE_NORM

GGML_ROPE_TYPE_NEOX

GGML_ROPE_TYPE_MROPE

GGML_ROPE_TYPE_VISION
```

Format

Integer constants

An object of class integer of length 1.

An object of class integer of length 1.

An object of class integer of length 1.

Details

Constants for RoPE (Rotary Position Embedding) modes used in transformer models. Different models use different RoPE implementations.

- `GGML_ROPE_TYPE_NORM` (0): Standard RoPE as in original paper (LLaMA, Mistral)
- `GGML_ROPE_TYPE_NEOX` (2): GPT-NeoX style RoPE with different interleaving
- `GGML_ROPE_TYPE_MROPE` (8): Multi-RoPE for multimodal models (Qwen2-VL)
- `GGML_ROPE_TYPE_VISION` (24): Vision model RoPE variant

Value

An integer constant representing a RoPE type

Examples

```
GGML_ROPE_TYPE_NORM    # 0 - Standard RoPE (LLaMA, Mistral)
GGML_ROPE_TYPE_NEOX    # 2 - GPT-NeoX style
GGML_ROPE_TYPE_MROPE   # 8 - Multi-RoPE (Qwen2-VL)
GGML_ROPE_TYPE_VISION  # 24 - Vision models
```

`RunGGML.SingleCellExperiment` *Run a GGML GPU operation on a Seurat object*

Description

High-level, Seurat-style entry point: extracts the expression matrix from the object, runs the requested operation on the GGML backend (Vulkan GPU with CPU fallback) and writes the result back into the object. Returns the updated object, so it slots into a `%>%` / `|>` pipeline next to `Seurat::RunPCA()`. Methods exist for `Seurat` and `SingleCellExperiment` objects (and a `.default` for bare matrices).

Usage

```
## S3 method for class 'SingleCellExperiment'
RunGGML(
  object,
  op = "embed",
  assay = NULL,
  layer = NULL,
  n_components = 50L,
  reduction_name = "ggml",
  device = "auto",
  genes = NULL,
  cells = NULL,
```

```
  reduction = NULL,
  dims = NULL,
  ...
)

RunGGML(
  object,
  op = "embed",
  assay = NULL,
  layer = NULL,
  n_components = 50L,
  reduction_name = "ggml",
  device = "auto",
  genes = NULL,
  cells = NULL,
  reduction = NULL,
  dims = NULL,
  ...
)

## Default S3 method:
RunGGML(
  object,
  op = "embed",
  assay = NULL,
  layer = NULL,
  n_components = 50L,
  reduction_name = "ggml",
  device = "auto",
  genes = NULL,
  cells = NULL,
  ...
)

## S3 method for class 'Seurat'
RunGGML(
  object,
  op = "embed",
  assay = NULL,
  layer = NULL,
  n_components = 50L,
  reduction_name = "ggml",
  device = "auto",
  genes = NULL,
  cells = NULL,
  reduction = NULL,
  dims = NULL,
  ...
```

)

Arguments

object	A Seurat or SingleCellExperiment object, or a bare feature-by-cell matrix/dgCMatrix (the .default method returns a ggml_result instead of an object).
op	Operation name; see ggml_ops_registry. Default "embed".
assay	Assay to read (Seurat); defaults to the object's default assay.
layer	Layer/slot to read; default "data".
n_components	Number of components for "embed" (PCA). Default 50.
reduction_name	Name of the reduction slot to create. Default "ggml".
device	"auto" (default), "vulkan" or "cpu".
genes, cells	Optional feature/cell subsets passed to extraction.
reduction	Optional name of an existing reduction to use as the input feature space (e.g. "pca" for "umap" / "neighbors"), instead of an assay layer. Seurat: Embeddings(); SCE: reducedDim().
dims	Optional integer vector selecting columns of reduction.
...	Additional parameters forwarded to the engine. A notable one is chunk_size: for op = "scale" and op = "embed" (PCA), passing an integer streams the sparse input in blocks of that many cells, densifying one block at a time so the full dense features-by-cells matrix (tens of GB at scale) is never held. Results are identical to the non-chunked path. op = "normalize" is already sparse, so chunk_size is a no-op there.

Details

Supported operations include "embed" (PCA), "normalize", "scale", "umap", "neighbors" and "largest_gene" (per-cell highest-expressed gene QC metric); see ggml_ops_registry.

Value

For a Seurat object, the updated object with a new reduction. For a bare matrix, a ggml_result.

Examples

```
## Not run:
library(Seurat)
pbmc <- RunGGML(pbmc, op = "embed", n_components = 30)
DimPlot(pbmc, reduction = "ggml")

## End(Not run)
```

summary.ggml_sequential_model *Summary method for ggml_sequential_model*

Description

Prints a detailed summary including input shape, layer details, trainable/non-trainable parameter counts, and memory estimate.

Usage

```
## S3 method for class 'ggml_sequential_model'
summary(object, ...)
```

Arguments

object	A ggml_sequential_model object
...	Additional arguments (ignored)

Value

The model object (invisibly).

tidy.ggmlr_parsnip_model *Tidy a fitted ggml parsnip model into a per-layer table*

Description

Returns one row per layer of the underlying sequential network, in broom style. Useful for comparing architectures across experiments in a R Markdown / Quarto report.

Usage

```
## S3 method for class 'ggmlr_parsnip_model'
tidy(x, ...)
```

Arguments

x	A fitted ‘ggmlr_parsnip_model‘ (the engine object inside a parsnip fit; e.g. from ‘extract_fit_engine()‘).
...	Unused; for generic compatibility.

Value

A [tibble][tibble::tibble] with columns: ‘layer‘ (name), ‘type‘, ‘units‘ (output units, ‘NA‘ if not applicable), ‘activation‘, ‘output_shape‘ (character), ‘params‘ (trainable parameter count) and ‘trainable‘ (logical).

Examples

```
ggml_set_n_threads(1L)  # deterministic, single OpenMP pool
spec <- parsnip::mlp(hidden_units = 8L, epochs = 3L) |>
  parsnip::set_engine("ggml", backend = "cpu") |>
  parsnip::set_mode("regression")
fit_obj <- parsnip::fit(spec, mpg ~ ., data = mtcars)
generics::tidy(parsnip::extract_fit_engine(fit_obj))
```

---

with_grad_tape *Run code with gradient tape enabled*

Description

Records all ag_* operations inside `expr` for later `backward()`. When the default device is `"gpu"`, the ggml context is reset at the start of each tape.

Usage

```
with_grad_tape(expr)
```

Arguments

`expr`	Expression to evaluate under gradient tape

Value

Value of last expression in expr (invisibly)

Inputs computed on the device

The reset frees every intermediate result of earlier `ag_*` calls; only `ag_param` weights (and tensors made from host matrices, which keep a host copy) survive it. An input built from `ag_*` operations on the GPU – an observation normalised with `ag_sub`/`ag_mul`, say – must therefore be built *inside* `expr`: built before the tape, it is destroyed on entry and the forward pass fails with "buffer was freed by a tape reset".

Examples

```
w <- ag_param(matrix(c(1, 0, 0, 1), 2, 2))
x <- ag_tensor(matrix(c(1, 2), 2, 1))
y <- ag_tensor(matrix(c(1, 2), 2, 1))
with_grad_tape({
  out  <- ag_matmul(w, x)
  loss <- ag_mse_loss(out, y)
})
backward(loss)
```

Index

∗ **backend**
ggml_backend_buffer_clear, 92
ggml_backend_buffer_get_usage, 94
ggml_backend_buffer_is_host, 95
ggml_backend_buffer_is_multi_buffer, 96
ggml_backend_buffer_reset, 97
ggml_backend_buffer_set_usage, 98
ggml_backend_buffer_usage_any, 99
ggml_backend_buffer_usage_compute, 100
ggml_backend_buffer_usage_weights, 101
ggml_backend_buft_get_alignment, 102
ggml_backend_buft_get_max_size, 103
ggml_backend_buft_is_host, 104
ggml_backend_buft_name, 105
ggml_backend_dev_buffer_type, 107
ggml_backend_dev_by_name, 108
ggml_backend_dev_by_type, 109
ggml_backend_dev_count, 110
ggml_backend_dev_description, 110
ggml_backend_dev_get, 111
ggml_backend_dev_get_props, 112
ggml_backend_dev_host_buffer_type, 113
ggml_backend_dev_init, 114
ggml_backend_dev_memory, 115
ggml_backend_dev_name, 116
ggml_backend_dev_offload_op, 117
ggml_backend_dev_supports_buft, 118
ggml_backend_dev_supports_op, 119
ggml_backend_dev_type, 120
ggml_backend_device_register, 121
ggml_backend_device_type_accel, 122
ggml_backend_device_type_cpu, 122
ggml_backend_device_type_gpu, 123
ggml_backend_device_type_igpu, 124
ggml_backend_event_free, 125
ggml_backend_event_new, 126
ggml_backend_event_record, 127
ggml_backend_event_synchronize, 128
ggml_backend_event_wait, 129
ggml_backend_get_device, 130
ggml_backend_graph_compute_async, 132
ggml_backend_graph_plan_compute, 133
ggml_backend_graph_plan_create, 134
ggml_backend_graph_plan_free, 135
ggml_backend_init_best, 136
ggml_backend_init_by_name, 137
ggml_backend_init_by_type, 138
ggml_backend_load, 139
ggml_backend_load_all, 140
ggml_backend_meta_device, 140
ggml_backend_multi_buffer_alloc_buffer, 142
ggml_backend_multi_buffer_set_usage, 143
ggml_backend_reg_by_name, 145
ggml_backend_reg_count, 146
ggml_backend_reg_dev_count, 146
ggml_backend_reg_dev_get, 147
ggml_backend_reg_get, 148
ggml_backend_reg_name, 149
ggml_backend_register, 150
ggml_backend_synchronize, 160
ggml_backend_tensor_copy_async, 161
ggml_backend_tensor_get_async, 163
ggml_backend_tensor_set_async, 165

ggml_backend_unload, 166
* **callbacks**
ggml_callback_early_stopping, 172
ggml_schedule_cosine_decay, 451
ggml_schedule_reduce_on_plateau, 451
ggml_schedule_step_decay, 452
* **cpu_features**
ggml_cpu_features, 190
ggml_cpu_get_rvv_vlen, 191
ggml_cpu_get_sve_cnt, 191
ggml_cpu_has_amx_int8, 192
ggml_cpu_has_arm_fma, 192
ggml_cpu_has_avx, 193
ggml_cpu_has_avx2, 194
ggml_cpu_has_avx512, 195
ggml_cpu_has_avx512_bf16, 195
ggml_cpu_has_avx512_vbmi, 196
ggml_cpu_has_avx512_vnni, 197
ggml_cpu_has_avx_vnni, 194
ggml_cpu_has_bmi2, 197
ggml_cpu_has_dotprod, 198
ggml_cpu_has_f16c, 198
ggml_cpu_has_fma, 199
ggml_cpu_has_fp16_va, 200
ggml_cpu_has_llamafile, 200
ggml_cpu_has_matmul_int8, 201
ggml_cpu_has_neon, 201
ggml_cpu_has_riscv_v, 202
ggml_cpu_has_sme, 203
ggml_cpu_has_sse3, 203
ggml_cpu_has_ssse3, 204
ggml_cpu_has_sve, 205
ggml_cpu_has_vsx, 205
ggml_cpu_has_vxe, 206
ggml_cpu_has_wasm_simd, 206
* **datasets**
GGML_GLU_OP_REGLU, 273
GGML_N_TASKS_MAX, 341
ggml_pool_1d, 409
GGML_SORT_ORDER_ASC, 476
GGML_TYPE_F32, 507
ggml_upscale, 513
rope_types, 573
* **graph**
ggml_graph_view, 284
ggml_op_can_inplace, 351
* **logging**
ggml_abort_is_r_enabled, 80
ggml_log_is_r_enabled, 328
ggml_log_set_default, 329
ggml_log_set_r, 329
ggml_set_abort_callback_default, 454
ggml_set_abort_callback_r, 455
* **op_info**
ggml_get_unary_op, 271
ggml_op_desc, 352
ggml_op_name, 353
ggml_op_symbol, 353
ggml_unary_op_name, 511
* **optimization**
ggml_fit_opt, 235
ggml_fit_opt_gen, 237
ggml_fit_opt_multi, 239
ggml_opt_alloc, 354
ggml_opt_context_optimizer_type, 355
ggml_opt_dataset_data, 356
ggml_opt_dataset_free, 357
ggml_opt_dataset_get_batch, 357
ggml_opt_dataset_get_batch_head, 358
ggml_opt_dataset_init, 360
ggml_opt_dataset_labels, 361
ggml_opt_dataset_ndata, 362
ggml_opt_dataset_shuffle, 363
ggml_opt_dataset_weights, 364
ggml_opt_default_params, 365
ggml_opt_epoch, 366
ggml_opt_eval, 367
ggml_opt_fit, 368
ggml_opt_fit_multi, 369
ggml_opt_free, 371
ggml_opt_get_lr, 372
ggml_opt_grad_acc, 373
ggml_opt_init, 374
ggml_opt_init_for_fit, 375
ggml_opt_init_for_fit_multi, 376
ggml_opt_inputs, 377
ggml_opt_labels, 378
ggml_opt_loss, 379
ggml_opt_loss_i, 380
ggml_opt_loss_type_binary_cross_entropy, 381
ggml_opt_loss_type_cross_entropy,

382
ggml_opt_loss_type_huber, 382
ggml_opt_loss_type_mae, 383
ggml_opt_loss_type_mean, 384
ggml_opt_loss_type_mse, 385
ggml_opt_loss_type_sum, 386
ggml_opt_loss_type_weighted_mse, 386
ggml_opt_n_loss, 387
ggml_opt_ncorrect, 388
ggml_opt_optimizer_name, 389
ggml_opt_optimizer_type_adamw, 390
ggml_opt_optimizer_type_sgd, 390
ggml_opt_outputs, 391
ggml_opt_outputs_i, 392
ggml_opt_pred, 393
ggml_opt_prepare_alloc, 394
ggml_opt_reset, 395
ggml_opt_result_accuracy, 396
ggml_opt_result_free, 397
ggml_opt_result_init, 397
ggml_opt_result_loss, 398
ggml_opt_result_loss_i, 399
ggml_opt_result_n_loss, 400
ggml_opt_result_ndata, 401
ggml_opt_result_pred, 402
ggml_opt_result_reset, 402
ggml_opt_set_lr, 403
ggml_opt_static_graphs, 404
ggml_snapshot_free, 470
ggml_trainer, 502

∗ **quantization**
dequantize_row_iq2_xxs, 70
dequantize_row_mxfp4, 71
dequantize_row_nvfp4, 72
dequantize_row_q1_0, 73
dequantize_row_q2_K, 73
dequantize_row_q4_0, 74
dequantize_row_tq1_0, 75
ggml_quant_block_info, 417
iq2xs_free_impl, 538
iq2xs_init_impl, 539
iq3xs_free_impl, 539
iq3xs_init_impl, 540
quantize_iq2_xxs, 562
quantize_mxfp4, 563
quantize_nvfp4, 564
quantize_q1_0, 565
quantize_q2_K, 565
quantize_q4_0, 566
quantize_row_iq3_xxs_ref, 567
quantize_row_mxfp4_ref, 568
quantize_row_nvfp4_ref, 569
quantize_row_q1_0_ref, 569
quantize_row_q2_K_ref, 570
quantize_row_q4_0_ref, 571
quantize_row_tq1_0_ref, 572
quantize_tq1_0, 572

∗ **rope**
ggml_rope_ext_inplace, 435
ggml_rope_multi, 436
ggml_rope_multi_inplace, 439

∗ **softmax**
ggml_soft_max_ext_back_inplace, 473
ggml_soft_max_ext_inplace, 474

∗ **state-space**
ggml_gated_linear_attn, 254
ggml_gated_linear_attn_back, 255
ggml_rwkv_output, 442
ggml_rwkv_wkv6, 443
ggml_rwkv_wkv6_back, 444
ggml_rwkv_wkv7, 445
ggml_rwkv_wkv7_back, 446
ggml_ssm_conv, 480
ggml_ssm_conv_back, 481
ggml_ssm_scan, 482
ggml_ssm_scan_back, 483
ggml_ssm_scan_output, 484

∗ **tensor_layout**
ggml_are_same_stride, 87
ggml_can_repeat, 172
ggml_count_equal, 188
ggml_is_contiguous_0, 294
ggml_is_contiguous_1, 294
ggml_is_contiguous_2, 295
ggml_is_contiguous_channels, 295
ggml_is_contiguous_rows, 296
ggml_is_contiguously_allocated, 296

∗ **tensor**
ggml_are_same_layout, 86
ggml_get_op_params, 268
ggml_get_op_params_f32, 268
ggml_get_op_params_i32, 269
ggml_set_op_params, 461

ggml_set_op_params_f32, 461
ggml_set_op_params_i32, 462
∗ **type_system**
ggml_blck_size, 168
ggml_ftype_to_ggml_type, 250
ggml_is_quantized, 298
ggml_type_name, 509
ggml_type_sizef, 510
%*%,ANY,ggml_matrix-method *(ggml_matrix-class)*, 333
%*%,ggml_matrix,ANY-method *(ggml_matrix-class)*, 333
%*%,ggml_matrix,ggml_matrix-method *(ggml_matrix-class)*, 333

ag_add, 17
ag_argmax, 18
ag_batch_norm, 18, *39*
ag_buffer, 19, *21*, *23*, *29*, *35*
ag_buffer_free *(ag_buffer)*, 19
ag_buffer_read *(ag_buffer)*, 19
ag_buffer_write *(ag_buffer)*, 19
ag_capture, *19*, 21, *22–24*, *35*, *56*
ag_capture_free, *21*, 22, *24*
ag_capture_set, 22, *24*
ag_capture_step, *22*, 23
ag_checkpoint, 24
ag_clamp, 25
ag_concat_rows, 26
ag_cross_entropy_loss, 27
ag_dataloader, 27
ag_default_device, 28
ag_default_dtype, 28
ag_device, 29
ag_dropout, 29
ag_dtype, 30, *32*, *50*
ag_embedding, 30
ag_estimate_training_memory, 31, *50*
ag_eval, 32
ag_exp, 33
ag_flash_attention, 33, *49, 50*
ag_gather, 34
ag_get_rows, *19–21*, *23*, 35, *56*
ag_grad, 36
ag_gradcheck, 36
ag_graph_mode, 37, *43*
ag_layer_norm, 39
ag_linear, 40
ag_load_model, *40*, 41, *54*
ag_load_state_dict, 42, *58*
ag_local_mode, *38*, 43
ag_log, 43
ag_log_softmax, 44
ag_matmul, 45
ag_matmul_precision, *38*, *43*, 45
ag_max, 46
ag_mean, 47
ag_mse_loss, 47, *552*
ag_mul, 48
ag_multihead_attention, *33*, 48
ag_param, 50, *578*
ag_pow, 51
ag_relu, 51
ag_reshape, 52
ag_sample_categorical, 52
ag_sample_normal, 53
ag_save_model, *41, 42*, 54, *58*
ag_scale, 55
ag_sequential, 55
ag_sigmoid, 56
ag_slice_rows, *26*, 56
ag_softmax, 57
ag_softmax_cross_entropy_loss, 57
ag_state_dict, *42*, 58
ag_sub, 59
ag_sum, 59
ag_tanh, 60
ag_tape_memory, *31, 32*, *50*, 60
ag_tensor, 61
ag_to_device, 62
ag_train, 62
ag_transpose, 63
aperm, *319*
as.matrix.ag_tensor, 63
as.matrix.ggml_matrix, 64
as_gpu_matrix, 64, *332, 333*
augment.ggmlr_parsnip_model, 65

backward, *24*, *50*, *60*, 66

check_grad_anomaly, 66
clip_grad_norm, *23*, 67, *68, 69*
clip_grad_value, 68
compile.ggml_functional_model *(compile.ggml_sequential_model)*, 69
compile.ggml_sequential_model, 69

crossprod,ggml_matrix-method
(ggml_matrix-class), 333

dequantize_row_iq1_m
(dequantize_row_iq2_xxs), 70
dequantize_row_iq1_s
(dequantize_row_iq2_xxs), 70
dequantize_row_iq2_s
(dequantize_row_iq2_xxs), 70
dequantize_row_iq2_xs
(dequantize_row_iq2_xxs), 70
dequantize_row_iq2_xxs, 70, *72–75*, *418*, *538–540*, *563–573*
dequantize_row_iq3_s
(dequantize_row_iq2_xxs), 70
dequantize_row_iq3_xxs
(dequantize_row_iq2_xxs), 70
dequantize_row_iq4_nl
(dequantize_row_iq2_xxs), 70
dequantize_row_iq4_xs
(dequantize_row_iq2_xxs), 70
dequantize_row_mxfp4, 71, *71–75*, *418*, *538–540*, *563–573*
dequantize_row_nvfp4, *71*, 72, *72–75*, *418*, *538–540*, *563–573*
dequantize_row_q1_0, *71*, *72*, 73, *74*, *75*, *418*, *538–540*, *563–573*
dequantize_row_q2_K, *71*, *72*, 73, *73*, *75*, *418*, *538–540*, *563–573*
dequantize_row_q3_K
(dequantize_row_q2_K), 73
dequantize_row_q4_0, *71–73*, 74, *74*, *75*, *418*, *538–540*, *563–573*
dequantize_row_q4_1
(dequantize_row_q4_0), 74
dequantize_row_q4_K
(dequantize_row_q2_K), 73
dequantize_row_q5_0
(dequantize_row_q4_0), 74
dequantize_row_q5_1
(dequantize_row_q4_0), 74
dequantize_row_q5_K
(dequantize_row_q2_K), 73
dequantize_row_q6_K
(dequantize_row_q2_K), 73
dequantize_row_q8_0
(dequantize_row_q4_0), 74
dequantize_row_q8_K
(dequantize_row_q2_K), 73
dequantize_row_tq1_0, *71–74*, 75, *75*, *418*, *538–540*, *563–573*
dequantize_row_tq2_0
(dequantize_row_tq1_0), 75
dim,ggml_matrix-method
(ggml_matrix-class), 333
dp_train, 76

evaluate, 77

fit.ggml_functional_model
(fit.ggml_sequential_model), 78
fit.ggml_sequential_model, 78

ggml_abort_is_r_enabled, 80, *328*, *329*, *454*, *455*
ggml_abs, 80
ggml_abs_inplace, 81
ggml_add, 82
ggml_add1, 84
ggml_add_inplace, 83
ggml_add_rel_pos, 84
ggml_apply, 85, *89*, *91*, *181*, *182*, *185*, *186*
ggml_arange, 86
ggml_are_same_layout, 86, *268*, *269*, *461*, *462*
ggml_are_same_shape, 87
ggml_are_same_stride, 87, *173*, *189*, *294–297*
ggml_argmax, 88
ggml_argsort, 89
ggml_attention, 89, *301*, *303*
ggml_backend_alloc_ctx_tensors, 91
ggml_backend_buffer_clear, 92, *94–96*, *98–105*, *107–129*, *131–142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_buffer_free, 93
ggml_backend_buffer_get_size, 93
ggml_backend_buffer_get_usage, *92*, 94, *95*, *96*, *98–105*, *107–129*, *131–142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_buffer_is_host, *92*, *94*, 95, *96*, *98–105*, *107–129*, *131–142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_buffer_is_multi_buffer, *92*, *94*, *95*, 96, *98–105*, *107–129*, *131–142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_buffer_name, 97

ggml_backend_buffer_reset, 92, 94–96, 97, 99–105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buffer_set_usage, 92, 94–96, 98, 98–105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buffer_usage_any, 92, 94–96, 98, 99, 99–105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buffer_usage_compute, 92, 94–96, 98, 99, 100, 101–105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buffer_usage_weights, 92, 94–96, 98–100, 101, 102–105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buft_get_alignment, 92, 94–96, 98–101, 102, 103–105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buft_get_max_size, 92, 94–96, 98–102, 103, 104, 105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buft_is_host, 92, 94–96, 98–103, 104, 105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_buft_name, 92, 94–96, 98–104, 105, 107–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_cpu_init, 106
ggml_backend_cpu_set_n_threads, 106
ggml_backend_dev_buffer_type, 92, 94–96, 98–105, 107, 108–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_by_name, 92, 94–96, 98–105, 107, 108, 109–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_by_type, 92, 94–96, 98–105, 107, 108, 109, 110–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_count, 92, 94–96, 98–105, 107–109, 110, 111–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_description, 92, 94–96, 98–105, 107–109, 110, 110, 112–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_get, 92, 94–96, 98–105, 107–110, 111, 111, 113–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_get_props, 92, 94–96, 98–105, 107–111, 112, 112, 114–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_host_buffer_type, 92, 94–96, 98–105, 107–112, 113, 113, 115–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_init, 92, 94–96, 98–105, 107–113, 114, 114, 116–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_memory, 92, 94–96, 98–105, 107–114, 115, 115, 117–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_name, 92, 94–96, 98–105, 107–115, 116, 116, 118–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_offload_op, 92, 94–96, 98–105, 107–116, 117, 117–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_supports_buft, 92, 94–96, 98–105, 107–117, 118, 118–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_supports_op, 92, 94–96, 98–105, 107–118, 119, 120–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_dev_type, 92, 94–96, 98–105, 107–119, 120, 121–129, 131–142, 144–151, 161–163, 165, 167
ggml_backend_device_register, 92, 94–96, 98–105, 107–120, 121, 122–129,

131–140, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_device_type_accel, *92*, *94–96*, *98–105*, *107–121*, 122, *123–129*, *131–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_device_type_cpu, *92*, *94–96*, *98–105*, *107–121*, 122, *122*, *124–129*, *131–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_device_type_gpu, *92*, *94–96*, *98–105*, *107–122*, 123, *123*, *125–129*, *131, 132*, *134–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_device_type_igpu, *92*, *94–96*, *98–105*, *107–123*, 124, *124–129*, *131, 132*, *134–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_event_free, *92*, *94–96*, *98–105*, *107–124*, 125, *125–129*, *131, 132*, *134–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_event_new, *92*, *94–96*, *98–105*, *107–125*, 126, *127–129*, *131, 132*, *134–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_event_record, *92*, *94–96*, *98–105*, *107–126*, 127, *128, 129*, *131, 132*, *134–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_event_synchronize, *92*, *94–96*, *98–105*, *107–127*, 128, *129*, *131, 132*, *134–140*, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_event_wait, *92*, *94–96*, *98–105*, *107–128*, 129, *131, 132*, *134–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_free, 130
ggml_backend_get_device, *92*, *94–96*, *98–105*, *107–129*, 130, *132*, *134–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_graph_compute, 131
ggml_backend_graph_compute_async, *92*, *94–96*, *98–105*, *107–129*, *131*, 132, *134–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_graph_plan_compute, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, 133, *135–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_graph_plan_create, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, 134, *134–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_graph_plan_free, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134*, 135, *135–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_init_best, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134, 135*, 136, *137–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_init_by_name, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–136*, 137, *138–140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_init_by_type, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–137*, 138, *139, 140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_load, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–138*, 139, *140*, *142–151*, *161–163*, *165*, *167*
ggml_backend_load_all, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–139*, 140, *142–151*, *161–163*, *165*, *167*
ggml_backend_meta_device, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–139*, 140, *140*, *143–151*, *161–163*, *165*, *167*
ggml_backend_multi_buffer_alloc_buffer, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–140*, 142, *142*, *144–151*, *161–163*, *165*, *167*
ggml_backend_multi_buffer_set_usage, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–140*, *142*, 143, *143*, *145–151*, *161–163*, *165*, *167*
ggml_backend_name, 144
ggml_backend_reg_by_name, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–140*, *142–144*, 145, *146–151*, *161–163*, *165*, *167*
ggml_backend_reg_count, *92*, *94–96*,

98–105, *107–129*, *131, 132*, *134–140*, *142–145*, 146, *147–151*, *161–163*, *165*, *167*
ggml_backend_reg_dev_count, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–140*, *142–145*, 146, *146*, *148–151*, *161–163*, *165*, *167*
ggml_backend_reg_dev_get, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–140*, *142–146*, 147, *147*, *149–151*, *161–163*, *165*, *167*
ggml_backend_reg_get, *92*, *94–96*, *98–105*, *107–129*, *131, 132*, *134–140*, *142–147*, 148, *148*, *150, 151*, *161–163*, *165*, *167*
ggml_backend_reg_name, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–148*, 149, *149*, *151*, *161–163*, *165*, *167*
ggml_backend_register, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–149*, 150, *150*, *161–163*, *165*, *167*
ggml_backend_sched_alloc_graph, 151
ggml_backend_sched_free, 152
ggml_backend_sched_get_backend, 152
ggml_backend_sched_get_n_backends, 153
ggml_backend_sched_get_n_copies, 153
ggml_backend_sched_get_n_splits, 154
ggml_backend_sched_get_tensor_backend, 154
ggml_backend_sched_graph_compute, *154*, 155
ggml_backend_sched_graph_compute_async, 156
ggml_backend_sched_new, 156
ggml_backend_sched_reserve, *154*, 157
ggml_backend_sched_reset, 158
ggml_backend_sched_set_tensor_backend, 159
ggml_backend_sched_synchronize, 159
ggml_backend_sched_trace, 160
ggml_backend_synchronize, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–151*, 160, *162, 163*, *165*, *167*
ggml_backend_tensor_copy_async, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–151*, 161, *161*, *163*, *165*, *167*
ggml_backend_tensor_get_and_sync, 162
ggml_backend_tensor_get_async, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–151*, *161, 162*, 163, *165*, *167*
ggml_backend_tensor_get_data, 164
ggml_backend_tensor_get_f32_first, 164
ggml_backend_tensor_set_async, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–151*, *161–163*, 165, *167*
ggml_backend_tensor_set_data, 166
ggml_backend_unload, *92*, *94–96*, *98–105*, *107*, *109–129*, *131, 132*, *134–140*, *142–151*, *161–163*, *165*, 166
ggml_batch_norm, 167
ggml_blck_size, 168, *250*, *298*, *509, 510*
ggml_build_backward_expand, 168, *170, 171*, *242*, *278*, *280*, *283, 284*, *458*, *463*
ggml_build_forward_expand, 169, *169, 170*
ggml_build_forward_expand_grads, *168, 169*, 170, *279*, *283, 284*, *459*
ggml_callback_early_stopping, 172, *231*, *451, 452*
ggml_can_repeat, *88*, 172, *189*, *294–297*
ggml_cast, 173, *174*
ggml_cast_numeric, 174, *174*
ggml_ceil, 175
ggml_ceil_inplace, 175
ggml_clamp, *169*, 176, *214*
ggml_compile, *69, 70*, *216*, *233*, *415*, *503*
ggml_compile *(*ggml_compile.ggml_functional_model*)*, 177
ggml_compile.ggml_functional_model, 177
ggml_concat, 179
ggml_cont, 179, *321*
ggml_conv_1d, *173*, 180
ggml_conv_1d_dw, 181
ggml_conv_1d_layer, 181
ggml_conv_2d, 182
ggml_conv_2d_direct, 183
ggml_conv_2d_dw, 184
ggml_conv_2d_dw_direct, 185

ggml_conv_2d_layer, 185
ggml_conv_transpose_1d, 187
ggml_conv_transpose_2d_p0, 187
ggml_cos, 188
ggml_count_equal, *88*, *173*, 188, *294–297*
ggml_cpu_add, 189
ggml_cpu_features, 190, *191–207*
ggml_cpu_get_rvv_vlen, *190*, 191, *192–207*
ggml_cpu_get_sve_cnt, *190*, 191, *191–207*
ggml_cpu_has_amx_int8, *190*, *191*, 192, *192–207*
ggml_cpu_has_arm_fma, *190*, *191*, 192, *192–207*
ggml_cpu_has_avx, *190–192*, 193, *193–207*
ggml_cpu_has_avx2, *190–193*, 194, *194–207*
ggml_cpu_has_avx512, *190–194*, 195, *195–207*
ggml_cpu_has_avx512_bf16, *190–194*, 195, *195–207*
ggml_cpu_has_avx512_vbmi, *190–195*, 196, *196–207*
ggml_cpu_has_avx512_vnni, *190–196*, 197, *198–207*
ggml_cpu_has_avx_vnni, *190–193*, 194, *195–207*
ggml_cpu_has_bmi2, *190–196*, 197, *197–207*
ggml_cpu_has_dotprod, *190–197*, 198, *198–207*
ggml_cpu_has_f16c, *190–197*, 198, *198–207*
ggml_cpu_has_fma, *190–198*, 199, *199–207*
ggml_cpu_has_fp16_va, *190–199*, 200, *201–207*
ggml_cpu_has_llamafile, *190–199*, 200, *200–207*
ggml_cpu_has_matmul_int8, *190–200*, 201, *201–207*
ggml_cpu_has_neon, *190–200*, 201, *201–207*
ggml_cpu_has_riscv_v, *190–201*, 202, *202–207*
ggml_cpu_has_sme, *190–202*, 203, *204–207*
ggml_cpu_has_sse3, *190–202*, 203, *203–207*
ggml_cpu_has_ssse3, *190–203*, 204, *204–207*
ggml_cpu_has_sve, *190–204*, 205, *206*, *207*
ggml_cpu_has_vsx, *190–204*, 205, *205–207*
ggml_cpu_has_vxe, *190–205*, 206, *206*, *207*
ggml_cpu_has_wasm_simd, *190–205*, 206, *206*
ggml_cpu_mul, 207
ggml_cpy, *173*, *174*, 208
ggml_cross_entropy_loss, 208, *209*, *210*
ggml_cross_entropy_loss_back, 209, *209*
ggml_crossprod, 210, *332*
ggml_cumsum, 210
ggml_custom, 211, *212–214*, *341*
ggml_custom_inplace, 212, *212*, *214*, *341*
ggml_custom_ops, *212*, 213, *213*
ggml_cycles, 214
ggml_cycles_per_ms, 215
ggml_default_mlp, 215
ggml_dense, *181*, *182*, *185*, *186*, 217
ggml_diag, 218
ggml_diag_mask_inf, 218
ggml_diag_mask_inf_inplace, 219
ggml_diag_mask_zero, 220
ggml_div, 220
ggml_div_inplace, 221
ggml_dup, 221
ggml_dup_inplace, 222
ggml_dup_tensor, 223
ggml_element_size, 223
ggml_elu, 224
ggml_elu_inplace, 224
ggml_embedding, 225
ggml_estimate_memory, *32*, 225
ggml_evaluate, *78*, *178*, *231*
ggml_evaluate (ggml_evaluate.ggml_functional_model), 226
ggml_evaluate.ggml_functional_model, 226
ggml_exp, 228
ggml_exp_inplace, 228
ggml_extract, 229
ggml_fit, *78*, *79*, *216*, *233*, *234*, *247*, *502*, *503*, *505*, *506*
ggml_fit (ggml_fit.ggml_functional_model), 230
ggml_fit.ggml_functional_model, 230
ggml_fit_generator, *232*, 233
ggml_fit_opt, *233*, 235, *237–239*, *241*, *355–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_fit_opt_gen, *234–236*, 237, *241*, *355–367*, *369*, *371–374*, *376–405*,

470, *503*, *505*
ggml_fit_opt_multi, *236*, 239, *239*, *355–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_flash_attn_back, 242
ggml_flash_attn_ext, *242*, 243, *243*
ggml_flash_attn_ext_add_sinks, 244, *471*
ggml_flash_attn_ext_get_prec, 245, *246*
ggml_flash_attn_ext_set_prec, 245, *245*
ggml_floor, 246
ggml_floor_inplace, 246
ggml_free, 247
ggml_freeze_bn_stats, 247, *249*, *505*
ggml_freeze_weights, *178*, *247*, 248, *511*
ggml_ftype_to_ggml_type, *168*, 250, *298*, *509, 510*
ggml_gallocr_alloc_graph, 250, *253*
ggml_gallocr_free, 251
ggml_gallocr_get_buffer_size, 251
ggml_gallocr_new, 252, *253*
ggml_gallocr_new_buft, 253
ggml_gallocr_reserve, 253
ggml_gated_linear_attn, 254, *255*, *442*, *444–446*, *480, 481*, *483–485*
ggml_gated_linear_attn_back, *254*, 255, *442*, *444–446*, *480, 481*, *483–485*
ggml_geglu, 255
ggml_geglu_quick, 256
ggml_geglu_split, 257
ggml_gelu, 257, *258*
ggml_gelu_back, 258
ggml_gelu_erf, *258*, 259
ggml_gelu_inplace, 259
ggml_gelu_quick, *258*, 260
ggml_get_f32, 261
ggml_get_f32_nd, 262
ggml_get_first_tensor, 262
ggml_get_i32, 263
ggml_get_i32_nd, 263
ggml_get_layer, 264
ggml_get_max_tensor_size, 264
ggml_get_mem_size, 265
ggml_get_n_threads, 266
ggml_get_name, 266
ggml_get_next_tensor, 267
ggml_get_no_alloc, 267
ggml_get_op_params, *87*, 268, *269*, *461, 462*
ggml_get_op_params_f32, *87*, 268, *268, 269*, *461, 462*
ggml_get_op_params_i32, *87*, *268*, 269, *269*, *461, 462*
ggml_get_rel_pos, 269
ggml_get_rows, *214*, 270
ggml_get_rows_back, 271
ggml_get_unary_op, 271, *352, 353*, *511*
ggml_glu, 272
GGML_GLU_OP_GEGLU *(*GGML_GLU_OP_REGLU*)*, 273
GGML_GLU_OP_GEGLU_ERF *(*GGML_GLU_OP_REGLU*)*, 273
GGML_GLU_OP_GEGLU_QUICK *(*GGML_GLU_OP_REGLU*)*, 273
GGML_GLU_OP_REGLU, 273
GGML_GLU_OP_SWIGLU *(*GGML_GLU_OP_REGLU*)*, 273
GGML_GLU_OP_SWIGLU_OAI *(*GGML_GLU_OP_REGLU*)*, 273
ggml_glu_split, 274
ggml_graph_add_node, 274
ggml_graph_clear, 275
ggml_graph_compute, *169*, 275, *283*
ggml_graph_compute_with_ctx, 276
ggml_graph_cpy, 277
ggml_graph_dump_dot, 278
ggml_graph_dup, *277*, 278
ggml_graph_expand, *274, 275*, 279
ggml_graph_get_grad, *168, 169*, *171*, 280
ggml_graph_get_grad_acc *(*ggml_graph_get_grad*)*, 280
ggml_graph_get_tensor, 281
ggml_graph_n_nodes, 281
ggml_graph_node, 282
ggml_graph_overhead, 282
ggml_graph_print, 283
ggml_graph_reset, *169*, 283
ggml_graph_view, 284, *352*
ggml_group_norm, 285
ggml_group_norm_inplace, 285
ggml_gru, 286
ggml_hardsigmoid, 287
ggml_hardswish, 288
ggml_im2col, 288
ggml_init, 289
ggml_init_auto, 290
ggml_inject, 291
ggml_input, 292

ggml_is_available, 293
ggml_is_contiguous, 293
ggml_is_contiguous_0, *88*, *173*, *189*, 294, *295–297*
ggml_is_contiguous_1, *88*, *173*, *189*, 294, *294–297*
ggml_is_contiguous_2, *88*, *173*, *189*, *294*, 295, *295–297*
ggml_is_contiguous_channels, *88*, *173*, *189*, *294*, 295, *295–297*
ggml_is_contiguous_rows, *88*, *173*, *189*, *294, 295*, 296, *296, 297*
ggml_is_contiguously_allocated, *88*, *173*, *189*, *294, 295*, 296, *296*
ggml_is_permuted, 297
ggml_is_quantized, *168*, *250*, 298, *509, 510*
ggml_is_transposed, 298
ggml_l2_norm, 299
ggml_l2_norm_inplace, 300
ggml_layer_add, 300
ggml_layer_attention, *90, 91*, 301, *309*, *311*, *320*, *323, 324*
ggml_layer_batch_norm, *248*, 303, *316*, *321, 322*
ggml_layer_concatenate, 304
ggml_layer_conv_1d, *182*, 305
ggml_layer_conv_2d, *186*, 306
ggml_layer_custom, *176*, 307
ggml_layer_dense, *216, 217*, 308
ggml_layer_dropout, *216*, 309
ggml_layer_embedding, 311, *319, 320*
ggml_layer_flatten, 312, *321*, *323*
ggml_layer_global_average_pooling_2d, 313
ggml_layer_global_max_pooling_2d, 313
ggml_layer_gru, *311*, 314
ggml_layer_layer_norm, 315, *350*
ggml_layer_lstm, *90*, *302*, *311*, 316
ggml_layer_max_pooling_2d, 317
ggml_layer_permute, 318, *321*
ggml_layer_positional_embedding, *302*, 319
ggml_layer_reshape, 320
ggml_layer_rms_norm, *315, 316*, 321
ggml_layer_sequence_pooling, 322, *324*
ggml_layer_transformer_block, 323
ggml_leaky_relu, 325
ggml_load_model, 326, *512*
ggml_load_weights, 326
ggml_log, 327
ggml_log_inplace, 328
ggml_log_is_r_enabled, *80*, 328, *329*, *454, 455*
ggml_log_set_default, *80*, *328*, 329, *329*, *454, 455*
ggml_log_set_r, *80*, *328*, 329, *329*, *454, 455*
ggml_lstm, 330
ggml_marshal_model, 330, *512*
ggml_matmul, *64*, *210*, 331, *332, 333*, *491, 492*
ggml_matmul_f64, 332
ggml_matrix, *64*
ggml_matrix-class, 333
ggml_mean, 334
ggml_model, *50*, 335, *504*
ggml_model_backend, 336, *506*
ggml_model_sequential, *50*, *216*, 336
ggml_mul, 337
ggml_mul_inplace, 338
ggml_mul_mat, 338
ggml_mul_mat_id, 339
ggml_n_dims, 340
GGML_N_TASKS_MAX, 341
ggml_nbytes, 341
ggml_neg, 342
ggml_neg_inplace, 343
ggml_nelements, 343
ggml_new_f32, 344
ggml_new_i32, 345
ggml_new_tensor, 345
ggml_new_tensor_1d, 346
ggml_new_tensor_2d, 347
ggml_new_tensor_3d, 347
ggml_new_tensor_4d, 348
ggml_norm, 349, *350*
ggml_norm_back, 350
ggml_norm_inplace, 350
ggml_nrows, 351
ggml_op_can_inplace, *284*, 351
ggml_op_desc, *272*, 352, *353*, *511*
ggml_op_name, *272*, *352*, 353, *353*, *511*
GGML_OP_POOL_AVG *(*ggml_pool_1d*)*, 409
GGML_OP_POOL_MAX *(*ggml_pool_1d*)*, 409
ggml_op_symbol, *272*, *352*, 353, *353*, *511*
ggml_ops_registry, 354, *441, 442*, *491*, *576*
ggml_opt_alloc, *236*, *239*, *241*, 354, *355–367*, *369*, *371–374*, *376–405*,

470, *505*
ggml_opt_context_optimizer_type, *236*, *239*, *241*, 355, *355–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_data, *236*, *239*, *241*, *355*, 356, *357–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_free, *236*, *239*, *241*, *355*, *356*, 357, *358–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_get_batch, *236*, *239*, *241*, *355*, *356*, 357, *357*, *359–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_get_batch_head, *236*, *239*, *241*, *355–357*, 358, *358*, *360–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_init, *236*, *239*, *241*, *355–359*, 360, *361–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_labels, *236*, *239*, *241*, *355–360*, 361, *362–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_ndata, *236*, *239*, *241*, *355–361*, 362, *363–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_shuffle, *236*, *239*, *241*, *355–362*, 363, *364–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_dataset_weights, *236*, *239*, *241*, *355–363*, 364, *365–367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_default_params, *236*, *239*, *241*, *355–364*, 365, *366*, *367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_epoch, *236*, *239*, *241*, *355–365*, 366, *367*, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_eval, *236*, *239*, *241*, *355–366*, 367, *369*, *371–374*, *376–405*, *470*, *505*
ggml_opt_fit, *236*, *239*, *241*, *355–367*, 368, *371–374*, *376–405*, *470*, *505*
ggml_opt_fit_multi, *236*, *239*, *241*, *355–367*, 369, *369*, *372–374*, *376–405*, *470*, *505*
ggml_opt_free, *236*, *239*, *241*, *355–366*, *368*, *369*, 371, *371–374*, *376–405*, *470*, *505*
ggml_opt_get_lr, *236*, *239*, *241*, *355–366*, *368*, *369*, *371*, 372, *372–374*, *376–405*, *470*, *505*
ggml_opt_grad_acc, *236*, *239*, *241*, *355–366*, *368*, *369*, *371*, *372*, 373, *374*, *376–405*, *470*, *505*
ggml_opt_init, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–373*, 374, *376–405*, *470*, *505*
ggml_opt_init_for_fit, *236*, *238*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, 375, *377–405*, *470*, *505*
ggml_opt_init_for_fit_multi, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, 376, *376*, *378–405*, *470*, *505*
ggml_opt_inputs, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, *376*, 377, *377*, *379–405*, *470*, *505*
ggml_opt_labels, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, *376*, *377*, 378, *378–405*, *470*, *505*
ggml_opt_labels_i *(*ggml_opt_outputs_i*)*, 392
ggml_opt_loss, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, *376–378*, 379, *379–405*, *470*, *505*
ggml_opt_loss_i, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, *376–379*, 380, *381–405*, *470*, *505*
ggml_opt_loss_type_binary_cross_entropy, *236*, *239*, *241*, *355–366*, *368*, *369*, *371–374*, *376–380*, 381, *382–405*, *470*, *505*
ggml_opt_loss_type_cross_entropy, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–381*, 382, *383–405*, *470*, *505*
ggml_opt_loss_type_huber, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–381*, 382, *382*, *384–405*, *470*, *505*
ggml_opt_loss_type_mae, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–382*, 383, *383–405*, *470*, *505*
ggml_opt_loss_type_mean, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–383*, 384, *384–405*, *470*, *505*
ggml_opt_loss_type_mse, *236*, *239*, *241*,

355–365, *367–369*, *371–374*, *376–384*, 385, *386–405*, *470*, *505*
ggml_opt_loss_type_sum, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–385*, 386, *387–405*, *470*, *505*
ggml_opt_loss_type_weighted_mse, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–385*, 386, *386*, *388–405*, *470*, *505*
ggml_opt_loss_weights_i *(*ggml_opt_outputs_i*)*, 392
ggml_opt_n_loss, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–386*, 387, *387*, *389–405*, *470*, *505*
ggml_opt_ncorrect, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–387*, 388, *388–405*, *470*, *505*
ggml_opt_ncorrect_i *(*ggml_opt_outputs_i*)*, 392
ggml_opt_optimizer_name, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–388*, 389, *389–405*, *470*, *505*
ggml_opt_optimizer_type_adamw, *236*, *239*, *241*, *355–365*, *367–369*, *371–374*, *376–389*, 390, *391–405*, *470*, *505*
ggml_opt_optimizer_type_sgd, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–389*, 390, *390*, *392–405*, *470*, *505*
ggml_opt_outputs, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–390*, 391, *391*, *393–405*, *470*, *505*
ggml_opt_outputs_i, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–391*, 392, *392*, *394–405*, *470*, *505*
ggml_opt_pred, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–392*, 393, *393*, *395–405*, *470*, *505*
ggml_opt_pred_i *(*ggml_opt_outputs_i*)*, 392
ggml_opt_prepare_alloc, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–393*, 394, *394–405*, *470*, *505*
ggml_opt_reset, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–394*, 395, *395–405*, *470*, *505*
ggml_opt_result_accuracy, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–395*, 396, *397–405*, *470*, *505*
ggml_opt_result_accuracy_i *(*ggml_opt_result_loss_i*)*, 399
ggml_opt_result_free, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–396*, 397, *398–405*, *470*, *505*
ggml_opt_result_init, *236*, *239*, *242*, *355–363*, *365*, *367–369*, *371–373*, *375–396*, 397, *397*, *399–405*, *470*, *505*
ggml_opt_result_loss, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–397*, 398, *398*, *400–405*, *470*, *505*
ggml_opt_result_loss_i, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–398*, 399, *399*, *401–405*, *470*, *505*
ggml_opt_result_n_loss, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–399*, 400, *400–405*, *470*, *505*
ggml_opt_result_ndata, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–400*, 401, *401–405*, *470*, *505*
ggml_opt_result_pred, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–401*, 402, *403–405*, *470*, *505*
ggml_opt_result_reset, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–401*, 402, *402*, *404*, *405*, *470*, *505*
ggml_opt_set_lr, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–402*, 403, *403*, *405*, *470*, *505*
ggml_opt_static_graphs, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–403*, 404, *404*, *470*, *505*
ggml_opt_step_adamw, 405
ggml_out_prod, 406
ggml_pad, 407, *408*
ggml_pad_ext, 408
ggml_pad_reflect_1d, 408
ggml_permute, *319*, 409

ggml_pool_1d, 409
ggml_pool_2d, 411
ggml_pop_layer, 411
ggml_pp_dp_forward, 412
ggml_pp_forward, *412*, 413, *413*
ggml_predict, *178*, *504*
ggml_predict (ggml_predict.ggml_functional_model), 415
ggml_predict.ggml_functional_model, 415
ggml_predict_classes, 416
ggml_print_mem_status, 416
ggml_print_objects, 417
ggml_quant_block_info, *71–75*, 417, *538–540*, *563–573*
ggml_quantize_chunk, 418
ggml_quantize_free, 419
ggml_quantize_init, 419
ggml_quantize_requires_imatrix, 420
ggml_reglu, 420
ggml_reglu_split, 421
ggml_relu, *176*, 422
ggml_relu_inplace, 422
ggml_repeat, 423
ggml_repeat_back, 424
ggml_reset, 424
ggml_reshape_1d, 425
ggml_reshape_2d, 426
ggml_reshape_3d, 426
ggml_reshape_4d, 427
ggml_result, *291, 292*, 428, *441*, *576*
ggml_rms_norm, 428
ggml_rms_norm_back, *350*, 429
ggml_rms_norm_inplace, 430
ggml_roll, 430
ggml_rope, 431
ggml_rope_ext, 432
ggml_rope_ext_back, 434
ggml_rope_ext_inplace, 435, *437*, *440*
ggml_rope_inplace, 436
ggml_rope_multi, 436, *436*, *438–440*
ggml_rope_multi_back, 438
ggml_rope_multi_inplace, *436, 437*, 439
GGML_ROPE_TYPE_MROPE (rope_types), 573
GGML_ROPE_TYPE_NEOX (rope_types), 573
GGML_ROPE_TYPE_NORM (rope_types), 573
GGML_ROPE_TYPE_VISION (rope_types), 573
ggml_round, 440
ggml_round_inplace, 441
ggml_run, 441, *491*
ggml_rwkv_output, *254, 255*, 442, *443–446*, *480, 481*, *483–485*
ggml_rwkv_state, *443*
ggml_rwkv_state (ggml_rwkv_output), 442
ggml_rwkv_wkv6, *254, 255*, *442*, 443, *444–446*, *480, 481*, *483–485*
ggml_rwkv_wkv6_back, *254, 255*, *442*, 444, *444–446*, *480, 481*, *483–485*
ggml_rwkv_wkv7, *254, 255*, *442*, *444*, 445, *446*, *480, 481*, *483–485*
ggml_rwkv_wkv7_back, *254, 255*, *442*, *444*, *445*, 446, *480, 481*, *483–485*
ggml_save_model, *248*, *331*, 446
ggml_save_weights, 447
ggml_scale, 448
ggml_scale_bias, 449
ggml_scale_inplace, 449
GGML_SCALE_MODE_BICUBIC (ggml_upscale), 513
GGML_SCALE_MODE_BILINEAR (ggml_upscale), 513
GGML_SCALE_MODE_NEAREST (ggml_upscale), 513
ggml_scatter_elements, 450
ggml_schedule_cosine_decay, *172*, 451, *452*
ggml_schedule_reduce_on_plateau, *172*, 451, *451, 452*
ggml_schedule_step_decay, *172*, *451*, 452, *452*
ggml_set, 453
ggml_set_1d, 453
ggml_set_2d, 454
ggml_set_abort_callback_default, *80*, *328, 329*, 454, *455*
ggml_set_abort_callback_r, *80*, *328, 329*, *454*, 455
ggml_set_f32, 455
ggml_set_f32_nd, 456
ggml_set_i32, 457
ggml_set_i32_nd, 457
ggml_set_input, 458
ggml_set_loss, *169*, 458, *463*
ggml_set_n_threads, 459
ggml_set_name, 459

ggml_set_no_alloc, 460
ggml_set_omp_threads, 460
ggml_set_op_params, *87*, *268, 269*, 461, *462*
ggml_set_op_params_f32, *87*, *268, 269*, 461, *461, 462*
ggml_set_op_params_i32, *87*, *268, 269*, *461*, 462, *462*
ggml_set_output, 463
ggml_set_param, *168, 169*, *171*, *280*, *405, 406*, *458, 459*, 463
ggml_set_seed, 464
ggml_set_zero, 465
ggml_sgn, 465
ggml_sigmoid, 466
ggml_sigmoid_inplace, 467
ggml_silu, 467
ggml_silu_back, *258*, 468
ggml_silu_inplace, 468
ggml_sin, 469
ggml_snapshot_free, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–405*, 470, *504, 505*
ggml_soft_max, 471
ggml_soft_max_add_sinks, *244*, 471
ggml_soft_max_ext, *471*, 472
ggml_soft_max_ext_back, 473
ggml_soft_max_ext_back_inplace, 473, *474*
ggml_soft_max_ext_inplace, 474, *474*
ggml_soft_max_inplace, 475
ggml_softplus, 475
ggml_softplus_inplace, 476
GGML_SORT_ORDER_ASC, 476
GGML_SORT_ORDER_DESC *(*GGML_SORT_ORDER_ASC*)*, 476
ggml_sqr, 477
ggml_sqr_inplace, 478
ggml_sqrt, 478
ggml_sqrt_inplace, 479
ggml_ssm_conv, *169*, *254, 255*, *442*, *444–446*, 480, *481*, *483–485*
ggml_ssm_conv_back, *254, 255*, *442*, *444–446*, *480*, 481, *483–485*
ggml_ssm_scan, *254, 255*, *442*, *444–446*, *480, 481*, 482, *483–485*
ggml_ssm_scan_back, *254, 255*, *442*, *444–446*, *480, 481*, 483, *483*, *485*
ggml_ssm_scan_output, *254, 255*, *442*, *444–446*, *480–483*, 484, *484*
ggml_ssm_scan_state, *482, 483*
ggml_ssm_scan_state *(*ggml_ssm_scan_output*)*, 484
ggml_step, 485
ggml_sub, 486
ggml_sub_inplace, 486
ggml_sum, *458*, 487
ggml_sum_rows, 488
ggml_swiglu, 488
ggml_swiglu_split, 489
ggml_tanh, 490
ggml_tanh_inplace, 490
ggml_task, *354*, *441, 442*, 491
ggml_tcrossprod, *332*, 491
ggml_tensor_copy, 492
ggml_tensor_data_ptr, 492
ggml_tensor_nb, 493
ggml_tensor_num, 493
ggml_tensor_overhead, 494
ggml_tensor_set_f32_scalar, 494
ggml_tensor_shape, 495
ggml_tensor_type, 495
ggml_test, 496
ggml_test_adamw_steps, 496
ggml_test_backend_ops, 497, *497*
ggml_time_init, 498
ggml_time_ms, 499
ggml_time_us, 499
ggml_timestep_embedding, 500
ggml_top_k, 500
ggml_tp_dp_forward, *412–414*, 501
ggml_trainer, *236*, *239*, *242*, *355–359*, *361–363*, *365*, *367–369*, *371–373*, *375–405*, *470*, 502
ggml_training_history, *336*, 506
ggml_transpose, 507
GGML_TYPE_BF16 *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_F16 *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_F32, 507
GGML_TYPE_I32 *(*GGML_TYPE_F32*)*, 507
ggml_type_name, *168*, *250*, *298*, 509, *510*
GGML_TYPE_Q2_K *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_Q3_K *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_Q4_0 *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_Q4_1 *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_Q4_K *(*GGML_TYPE_F32*)*, 507
GGML_TYPE_Q5_K *(*GGML_TYPE_F32*)*, 507

GGML_TYPE_Q6_K (GGML_TYPE_F32), 507
GGML_TYPE_Q8_0 (GGML_TYPE_F32), 507
ggml_type_size, 510
ggml_type_sizef, *168*, *250*, *298*, *509*, 510
ggml_unary_op_name, *272*, *352, 353*, 511
ggml_unfreeze_bn_stats (ggml_freeze_bn_stats), 247
ggml_unfreeze_weights, 511
ggml_unmarshal_model, *331*, 512
ggml_upscale, 513
ggml_used_mem, 514
ggml_version, 515
ggml_view_1d, 515
ggml_view_2d, 516
ggml_view_3d, 516
ggml_view_4d, 517
ggml_view_tensor, 518
ggml_vulkan_available, 518
ggml_vulkan_backend_name, 519
ggml_vulkan_device_caps, 519
ggml_vulkan_device_count, 520
ggml_vulkan_device_description, 520
ggml_vulkan_device_groups, 521
ggml_vulkan_device_memory, 521
ggml_vulkan_free, 522
ggml_vulkan_hard_exit_available, 523, *528*
ggml_vulkan_init, 523
ggml_vulkan_is_backend, 524
ggml_vulkan_list_devices, 525, *532*
ggml_vulkan_p2p_selftest, 525, *530*
ggml_vulkan_shutdown, *523*, 527
ggml_vulkan_split_buffer_type, 528
ggml_vulkan_split_mul_mat, *413*, *501, 502*, 529
ggml_vulkan_split_row_ranges, *528*, *530*, 531
ggml_vulkan_status, 532
ggml_win_part, 532, *533*
ggml_win_unpart, 533
ggml_with_temp_ctx, 534
gguf_free, 534
gguf_load, 535, *535*
gguf_metadata, 535
gguf_tensor_data, *535*, 536
gguf_tensor_info, 536
gguf_tensor_names, 537
glance.ggmlr_parsnip_model, 537

iq2xs_free_impl, *71–75*, *418*, 538, *539, 540*, *563–573*
iq2xs_init_impl, *71–75*, *418*, *538*, 539, *540*, *563–573*
iq3xs_free_impl, *71–75*, *418*, *538*, 539, *539, 540*, *563–573*
iq3xs_init_impl, *71–75*, *418*, *538, 539*, 540, *540*, *563–573*

lr_scheduler_cosine, 541
lr_scheduler_cyclic, 542
lr_scheduler_onecycle, 543
lr_scheduler_step, 544
lr_scheduler_warmup_cosine, 545

nn_topo_sort, 546

onnx_device_info, 546
onnx_graph_info, 547
onnx_inputs, 548
onnx_load, 548, *554*
onnx_run, 549, *554*
onnx_summary, 550
optimizer_adam, *23*, *406*, 550
optimizer_sgd, 551, *551*

plot.ggml_history, 552
predict, *553*
predict.ggml_functional_model (predict.ggml_sequential_model), 553
predict.ggml_sequential_model, 553
predict.onnx_model, *549*, 554
print.ag_tensor, 555
print.ggml_functional_model, 555
print.ggml_history, 556
print.ggml_sequential_model, 556
print.onnx_model, 557
pt_cube4_layout, 557, *559, 560*
pt_forward, 558, *559*
pt_transformer, *557*, 559, *560*
pt_transformer_load, *558, 559*, 560
pth_catalogue, 560, *560–562*
pth_load, 561

quantize_iq1_m (quantize_iq2_xxs), 562
quantize_iq1_s (quantize_iq2_xxs), 562
quantize_iq2_s (quantize_iq2_xxs), 562
quantize_iq2_xs (quantize_iq2_xxs), 562

quantize_iq2_xxs, *71–75*, *418*, *538–540*, 562, *564–573*
quantize_iq3_s *(quantize_iq2_xxs)*, 562
quantize_iq3_xxs *(quantize_iq2_xxs)*, 562
quantize_iq4_nl *(quantize_iq2_xxs)*, 562
quantize_iq4_xs *(quantize_iq2_xxs)*, 562
quantize_mxfp4, *71–75*, *418*, *538–540*, 563, *563–573*
quantize_nvfp4, *71–75*, *418*, *538–540*, *563*, 564, *564–573*
quantize_q1_0, *71–75*, *418*, *538–540*, *563*, *564*, 565, *566–573*
quantize_q2_K, *71–75*, *418*, *538–540*, *563*, *564*, 565, *565*, *567–573*
quantize_q3_K *(quantize_q2_K)*, 565
quantize_q4_0, *71–75*, *418*, *538–540*, *563–565*, 566, *566*, *568–573*
quantize_q4_1 *(quantize_q4_0)*, 566
quantize_q4_K *(quantize_q2_K)*, 565
quantize_q5_0 *(quantize_q4_0)*, 566
quantize_q5_1 *(quantize_q4_0)*, 566
quantize_q5_K *(quantize_q2_K)*, 565
quantize_q6_K *(quantize_q2_K)*, 565
quantize_q8_0 *(quantize_q4_0)*, 566
quantize_row_iq2_s_ref *(quantize_row_iq3_xxs_ref)*, 567
quantize_row_iq3_s_ref *(quantize_row_iq3_xxs_ref)*, 567
quantize_row_iq3_xxs_ref, *71–75*, *418*, *538–540*, *563–566*, 567, *567–573*
quantize_row_iq4_nl_ref *(quantize_row_iq3_xxs_ref)*, 567
quantize_row_iq4_xs_ref *(quantize_row_iq3_xxs_ref)*, 567
quantize_row_mxfp4_ref, *71–75*, *418*, *538–540*, *563–567*, 568, *568–573*
quantize_row_nvfp4_ref, *71–75*, *418*, *538–540*, *563–568*, 569, *570–573*
quantize_row_q1_0_ref, *71–75*, *418*, *538–540*, *563–568*, 569, *569*, *571–573*
quantize_row_q2_K_ref, *71–75*, *418*, *538–540*, *563–569*, 570, *570–573*
quantize_row_q3_K_ref *(quantize_row_q2_K_ref)*, 570
quantize_row_q4_0_ref, *71–75*, *418*, *538–540*, *563–570*, 571, *571–573*
quantize_row_q4_1_ref *(quantize_row_q4_0_ref)*, 571
quantize_row_q4_K_ref *(quantize_row_q2_K_ref)*, 570
quantize_row_q5_0_ref *(quantize_row_q4_0_ref)*, 571
quantize_row_q5_1_ref *(quantize_row_q4_0_ref)*, 571
quantize_row_q5_K_ref *(quantize_row_q2_K_ref)*, 570
quantize_row_q6_K_ref *(quantize_row_q2_K_ref)*, 570
quantize_row_q8_0_ref *(quantize_row_q4_0_ref)*, 571
quantize_row_q8_1_ref *(quantize_row_q4_0_ref)*, 571
quantize_row_q8_K_ref *(quantize_row_q2_K_ref)*, 570
quantize_row_tq1_0_ref, *71–75*, *418*, *538–540*, *563–571*, 572, *573*
quantize_row_tq2_0_ref *(quantize_row_tq1_0_ref)*, 572
quantize_tq1_0, *71–75*, *418*, *538–540*, *563–571*, 572, *572*
quantize_tq2_0 *(quantize_tq1_0)*, 572

rope_types, 573
RunGGML *(RunGGML.SingleCellExperiment)*, 574
RunGGML.SingleCellExperiment, 574

set.seed, *464*
show,ggml_matrix-method *(ggml_matrix-class)*, 333
summary.ggml_sequential_model, 577

tcrossprod,ggml_matrix-method *(ggml_matrix-class)*, 333
tidy.ggmlr_parsnip_model, 577

warning, *523*, *528*
with_grad_tape, *60*, 578