Package: ggmlR (via r-universe)

June 4, 2026

Type Package

Title 'GGML' Tensor Operations for Machine Learning

Version 0.7.8

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

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

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/build/clean-inst-doc false

NeedsCompilation yes

Author Yuri Baramykov [aut, cre] (ORCID:
<https://orcid.org/0009-0000-7627-4217>), Georgi Gerganov [ctb, cph] (Author of the GGML library), Jeffrey Quesnelle [ctb, cph] (Contributor to ops.cpp), Bowen Peng [ctb, cph] (Contributor to ops.cpp), Mozilla Foundation [ctb, cph] (Author of llamafile/sgemm.cpp)

Maintainer Yuri Baramykov <lbsbmsu@mail.ru>

Config/pak/sysreqs make

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

Date/Publication 2026-06-04 06:59:06 UTC

RemoteUrl https://github.com/cran/ggmlR

RemoteRef HEAD

RemoteSha 4adf9d7079180282c9348c80c6ca7472b08fade3

Contents

---

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_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_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_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"`

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. Backward pass always uses f32 R matrices regardless of this setting.

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_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_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_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)
```

Arguments

in_features	Input dimension
out_features	Output dimension
activation	"relu", "sigmoid", "tanh", "softmax", or NULL

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)
```

ag_log *Element-wise natural logarithm*

Description

Element-wise natural logarithm

Usage

```
ag_log(x)
```

Arguments

x	ag_tensor

Value

ag_tensor

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_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()

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_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_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_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_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

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)
```

---

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`	Named list of ag_param tensors (same as passed to optimizer).
`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)
```

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" or "mse".
metrics	Character vector of metrics (default "accuracy").
...	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_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_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_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_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_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_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_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.ggml_sequential_model` *Evaluate a Model*

Description

Computes loss and metrics on test data. This is the keras-compatible interface; it delegates to `ggml_evaluate`.

Usage

```
## S3 method for class 'ggml_sequential_model'
evaluate(x, test_x, test_y, batch_size = 32L, ...)

## S3 method for class 'ggml_functional_model'
evaluate(x, test_x, test_y, batch_size = 32L, ...)
```

Arguments

`x`	A trained model object.
`test_x`	Test data.
`test_y`	Test labels.
`batch_size`	Batch size (default 32).
`...`	Additional arguments passed to `ggml_evaluate`.

Value

A named list with `loss` and metric values.

`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,
  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,
  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).

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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_dev_by_name(), 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_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_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_dev_by_name(), ggml_backend_dev_by_type(), 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_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_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_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), 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_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_dev_by_name(), ggml_backend_dev_by_type(), ggml_backend_dev_count(), ggml_backend_dev_description(), 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_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_dev_by_name()`, `ggml_backend_dev_by_type()`, `ggml_backend_dev_count()`,
`ggml_backend_dev_description()`, `ggml_backend_dev_get()`, `ggml_backend_dev_init()`, `ggml_backend_dev_memor`
`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_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_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_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_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_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_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_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_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_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_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_
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_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_`
`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_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_
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_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_`
`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_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_
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_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_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_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_
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_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_
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_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_
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_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_ 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_weight, 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_, 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_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_`
`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_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_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_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_
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_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_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_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_
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_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_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_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_
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_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_
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_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_
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_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_
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_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_
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_weigh
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_
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_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_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_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_
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_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_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_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_
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_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_
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_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_
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_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_`
`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_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_
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_se
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_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_
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_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_
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_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_
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_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_
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_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_
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_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_
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_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_

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_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_ 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

Value

Backend pointer or NULL if not assigned

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_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_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_
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_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_
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_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_
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_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_
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)

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_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_
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_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_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_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]

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"
)

ggml_compile(
  model,
  optimizer = "adam",
```

```
  loss = "categorical_crossentropy",
  metrics = c("accuracy"),
  backend = "auto"
)

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

Arguments

model	A ggml_sequential_model object
optimizer	Optimizer name: "adam" or "sgd"
loss	Loss function name: "categorical_crossentropy" or "mse"
metrics	Character vector of metrics (currently "accuracy")
backend	Backend to use: "auto" (GPU if available, else CPU), "cpu", or "vulkan"

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_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_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_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)
```

Arguments

units	Number of output units.
activation	Activation function name or NULL.
name	Optional character name.
trainable	Logical; whether weights are updated during training.

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`.

Examples

```
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_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,
  ...
)

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).
`...`	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 Numeric vector of per-sample weights (length = nrow(x)). Cannot be used with class_weight.

verbose 0 = silent, 1 = progress (default: 1)

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)

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

```
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_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,
  callbacks = list(),
  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)
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

Value

Data frame with columns epoch, train_loss, train_accuracy, val_loss, val_accuracy

See Also

Other optimization: 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_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_type_cross_entropy(), 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_typ 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_flash_attn_back *Flash Attention Backward (Graph)*

Description

Backward pass for Flash Attention. Used during training to compute gradients through attention.

Usage

```
ggml_flash_attn_back(ctx, q, k, v, d, masked = TRUE)
```

Arguments

ctx	GGML context
q	Query tensor (same as forward pass)
k	Key tensor (same as forward pass)
v	Value tensor (same as forward pass)
d	Gradient tensor from upstream (same shape as forward output)
masked	Logical: whether causal masking was used in forward pass

Value

Gradient tensor

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_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_weights *Freeze Layer Weights*

Description

Sets `trainable = FALSE` on layers, preventing their weights from being updated during training. Accepts optional `from` / `to` to freeze a range of layers by index, or `layer_names` to freeze by name. If none are provided, all layers are frozen.

Usage

```
ggml_freeze_weights(
  model,
  from = 1L,
  to = length(model$layers),
  layer_names = 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)
...	Additional arguments passed to methods

Value

The model with selected layers frozen.

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_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_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_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_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_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_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

Resets the computation graph for a new backward pass. NOTE: This function requires the graph to have gradients allocated (used for training/backpropagation). For inference-only graphs, this function will cause an error.

Usage

```
ggml_graph_reset(graph)
```

Arguments

graph	Graph object with gradients allocated

Value

No return value, called for side effects

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_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_allocat

---

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_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 *Create a Conv1D Layer Object*

Description

Create a Conv1D Layer Object

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
)

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

A ggml_layer object.

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` *Create a Conv2D Layer Object*

Description

Create a Conv2D Layer Object

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
)

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

A ggml_layer object.

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_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
)
```

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.

Value

The model object with the dense 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_dense(128, activation = "relu")
```

---

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)
```

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.

Value

The model with the embedding layer appended, or a new tensor node.

Axis order (ggml vs Keras)

ggml stores tensors in column-major order, so the output shape is `[dim, seq_len]` per sample (ggml convention) rather than `[seq_len, dim]` as in Keras. When you call `ggml_layer_flatten()` after embedding the result is the same flattened vector regardless of order, but if you access raw output tensors be aware of this transposition.

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_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_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_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_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_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_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_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_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_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_type_cross_entropy(), 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_typ 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_opt_alloc(), 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_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_type_cross_entropy(), 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_typ 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_opt_alloc(), ggml_opt_context_optimizer_type(), 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_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_type_cross_entropy(), 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_typ
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_ndata(),
ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

`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_opt_alloc(), ggml_opt_context_optimizer_type(),
ggml_opt_dataset_data(), 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_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_type_cross_entropy(), 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_typ
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_ndata(),
ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_opt_alloc(), ggml_opt_context_optimizer_type(), ggml_opt_dataset_data(), ggml_opt_dataset_free(), 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_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_type_cross_entropy(), 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_typ 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

`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_opt_alloc()`, `ggml_opt_context_optimizer_type()`, `ggml_opt_dataset_data()`, `ggml_opt_dataset_free()`, `ggml_opt_dataset_get_batch()`, `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_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_type_cross_entropy()`, `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_typ`
`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_ndata()`, `ggml_opt_result_pred()`, `ggml_opt_result_reset()`, `ggml_opt_set_lr()`, `ggml_opt_static_graphs()`

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_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_ndata(), ggml_opt_dataset_shuffle(), ggml_opt_dataset_weights(), 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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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(), ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_shuffle(), ggml_opt_dataset_weights(), 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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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(), ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_weights(), ggml_opt_default_param, 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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

`ggml_opt_dataset_weights` *Get dataset per-datapoint weights tensor*

Description

Returns the (lazily allocated) per-datapoint weights tensor with shape [1, 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)
```

Arguments

`dataset`	External pointer to dataset

Value

External pointer to weights tensor

See Also

Other optimization: ggml_fit_opt(), 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_default_param, 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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_weigh, 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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_weigh, ggml_opt_default_params(), 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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_weigh`, `ggml_opt_default_params()`, `ggml_opt_epoch()`, `ggml_opt_fit()`, `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_type_cross_entropy()`, `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_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_ndata()`, `ggml_opt_result_pred()`, `ggml_opt_result_reset()`, `ggml_opt_set_lr()`, `ggml_opt_static_graphs()`

`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_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_weigh, ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), 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_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_weigh, ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), 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_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_weigh, ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_free(), ggml_opt_grad_acc(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_weigh, ggml_opt_default_params(), ggml_opt_epoch(), ggml_opt_eval(), ggml_opt_fit(), ggml_opt_free(), ggml_opt_get_lr(), ggml_opt_init(), ggml_opt_init_for_fit(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_for_fit(), ggml_opt_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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
)
```

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)

Value

List with elements 'opt_ctx' and 'lr_ud'

See Also

Other optimization: ggml_fit_opt(), 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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_mean(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), 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_accuracy(), ggml_opt_result_free(), ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), ggml_opt_loss_type_mean(), ggml_opt_loss_type_mse(), ggml_opt_loss_type_sum(), ggml_opt_loss_type_weighted_mse(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_sgd(), ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

`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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

`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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, ggml_opt_outputs(), 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, ggml_opt_outputs(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accurac, ggml_opt_result_init(), ggml_opt_result_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, ggml_opt_outputs(), ggml_opt_pred(), ggml_opt_prepare_alloc(), ggml_opt_reset(), ggml_opt_result_accurac

ggml_opt_result_free(), ggml_opt_result_loss(), ggml_opt_result_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_pred(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

---

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_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_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_inputs(),

ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ 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_ndata(), ggml_opt_result_reset(), ggml_opt_set_lr(), ggml_opt_static_graphs()

`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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ 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_ndata(), ggml_opt_result_pred(), ggml_opt_set_lr(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_static_graphs()

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_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_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_inputs(), ggml_opt_labels(), ggml_opt_loss(), ggml_opt_loss_type_cross_entropy(), 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_typ, 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_ndata(), ggml_opt_result_pred(), ggml_opt_result_reset(), ggml_opt_set_lr()

---

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_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_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, ...)

ggml_predict(model, ...)

## S3 method for class 'ggml_sequential_model'
ggml_predict(model, x, batch_size = 32L, ...)
```

Arguments

model	A trained ggml_sequential_model
x	Input data (matrix or array)
batch_size	Batch size for inference
...	Additional arguments (ignored).

Value

Matrix of predictions with shape [N, output_units]

---

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_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_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	Input tensor (x from forward pass)
b	Gradient tensor (dy)
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_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_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_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_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_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)

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_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_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_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_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_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_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_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.

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. Accepts optional from / to to unfreeze a range of layers, or layer_names to unfreeze by name. If none are provided, all layers are unfrozen.

Usage

```
ggml_unfreeze_weights(
  model,
  from = 1L,
  to = length(model$layers),
  layer_names = 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)
...	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_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_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_status *Print Vulkan status*

Description

Prints information about Vulkan availability and devices.

Usage

```
ggml_vulkan_status()
```

Value

NULL (invisible), prints status to console

Examples

```
ggml_vulkan_status()
```

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.

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_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_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_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_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)
```

Arguments

optimizer	Optimizer environment.
T_max	Number of steps for one cosine cycle.
lr_min	Minimum learning rate (default 0).
restart	Logical; if TRUE restart after T_max steps.

Value

An `lr_scheduler_cosine` environment

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()
```

---

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
```

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_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 with dim attributes for multi-dimensional outputs).

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)
```

Arguments

params	Named list of ag_param tensors
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)

Value

An optimizer environment

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)
```

Arguments

params	Named list of ag_param tensors
lr	Learning rate (default 0.01)
momentum	Momentum factor (default 0)

Value

An optimizer environment

Examples

```
w <- ag_param(matrix(runif(4), 2, 2))
opt <- optimizer_sgd(list(w = w), lr = 0.01)
```

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 = 32L, ...)

## S3 method for class 'ggml_functional_model'
predict(object, x, batch_size = 32L, ...)
```

Arguments

`object`	A trained model object.
`x`	Input data (matrix, array, or list for multi-input models).
`batch_size`	Batch size for inference (default 32).
`...`	Additional arguments (ignored).

Value

Matrix of predictions.

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.

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_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_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_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_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_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_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_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_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_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_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_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_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
```

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).

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)

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