Package: Rllm (via r-universe)

July 10, 2026

Title Native Quantized Matrix Products and LLM Inference over 'Rfmalloc' Tensors

Version 0.1.0

Description The composition layer of the 'Rfmalloc' ecosystem: it registers 'Rggml' (a vendored 'GGML' build with runtime-SIMD-dispatched quantized kernels) as a codec-aware matrix-multiply backend for 'Rfmalloc', so that products of file-backed, quantized tensors run natively in quantized space (weights stay 'Q4_K'/'Q6_K'/... encoded, activations are quantized on the fly) instead of being decoded to double first. Combined with 'Rgguf', which exposes 'GGUF' model weights as 'Rfmalloc'-backed tensors, this lets a larger-than-memory model's linear layers be multiplied through 'GGML's SIMD-accelerated dot kernels zero-copy from the memory-mapped payload. A transformer forward-pass graph builder over these primitives is planned.

License GPL (>= 2)

Encoding UTF-8

Imports Rfmalloc, Rgguf, Rggml

LinkingTo Rfmalloc, Rggml

Suggests tinytest

Remotes sounkou-bioinfo/Rfmalloc/packages/Rfmalloc, sounkou-bioinfo/Rfmalloc/packages/Rgguf, sounkou-bioinfo/Rfmalloc/packages/Rggml

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.4.0)

URL https://github.com/sounkou-bioinfo/Rfmalloc, https://sounkou-bioinfo.github.io/Rfmalloc/Rllm/

BugReports https://github.com/sounkou-bioinfo/Rfmalloc/issues

Repository https://sounkou-bioinfo.r-universe.dev

Date/Publication 2026-07-10 06:12:39 UTC

RemoteUrl https://github.com/sounkou-bioinfo/Rfmalloc

RemoteRef HEAD

RemoteSha 921d01f6786a2db38d90a2629c39f56a244b7875

RemoteSubdir packages/Rllm

Contents

`rllm_decode` *Convert between token ids and raw bytes*

Description

The ids <-> bytes edge codecs of the bytes-boundary API, built from the model's own GGUF tokenizer metadata (byte-level BPE, `tokenizer.ggml.model == "gpt2"`). `rllm_decode()` maps ids to the exact bytes they stand for. `rllm_encode()` byte-pair-encodes bytes into ids using the file's merge ranks; it applies the merges without GPT-2's regex pre-tokenizer, so a tokenization may occasionally differ from llama.cpp's canonical split - every output is still a valid encoding of the input (`rllm_decode(rllm_encode(x))` is always `x`).

Usage

```
rllm_decode(model, ids)

rllm_encode(model, x)
```

Arguments

`model`	An `rllm_model` whose GGUF carries a byte-level BPE tokenizer.
`ids`	Integer vector of 0-based token ids.
`x`	A raw vector (or a single string, converted with `charToRaw()`).

Value

`rllm_decode()`: a raw vector. `rllm_encode()`: an integer vector of 0-based token ids.

rllm_forward *Run a transformer forward pass and return the logits*

Description

Assembles the GGML compute graph for a llama-architecture forward pass (RMSNorm, RoPE, causal self-attention, SwiGLU feed-forward) over the model's memory-mapped weights and computes it on the GGML CPU backend. Quantized weights are contracted natively through the SIMD-dispatched quantized kernels - they are never decoded to double.

Usage

```
rllm_forward(model, tokens, cache = NULL)
```

Arguments

model	An rllm_model from rllm_gguf_model().
tokens	Integer vector of 0-based token ids (as in the GGUF vocab).
cache	Optional rllm_kv_cache() for incremental decoding.

Details

Without a cache, the graph attends over the whole token batch with a causal mask (prompt scoring). With a rllm_kv_cache(), the pass appends the new tokens' keys/values to the cache and attends over everything cached so far, advancing cache$n_past - the incremental-decoding path: prefill once with the prompt, then feed one token at a time.

Value

A numeric matrix of logits, dim c(n_vocab, length(tokens)): column i scores the token following position i.

rllm_generate *Generate tokens with a KV cache (greedy or sampled)*

Description

The bytes-in/bytes-out generation entry point: prefills a KV cache with the prompt (ids or raw bytes), then decodes one token per step - each step is a single-token rllm_forward() over the cache, not a full re-forward. Stops after n_new tokens or at the model's EOS token. Decoding is greedy by default (temperature = 0); set a positive temperature for temperature-scaled sampling, optionally narrowed by top_k and/or top_p.

Usage

```
rllm_generate(
  model,
  prompt,
  n_new = 32L,
  temperature = 0,
  top_k = 0L,
  top_p = 1,
  seed = NULL,
  cache = NULL,
  runtime = NULL
)
```

Arguments

model	An rllm_model from rllm_gguf_model().
prompt	Integer vector of 0-based token ids, or a raw vector (encoded with rllm_encode(); a single string is converted via charToRaw() as a convenience).
n_new	Maximum number of tokens to generate.
temperature	Sampling temperature. 0 (default) is greedy/argmax; larger values flatten the distribution (more diverse).
top_k	Keep only the top_k highest-logit tokens before sampling (0, default, disables the cutoff). Ignored when greedy.
top_p	Nucleus sampling: keep the smallest set of tokens whose probabilities sum to at least top_p (1, default, disables it). Ignored when greedy.
seed	Optional integer seed, for reproducible sampling.
cache	Optional rllm_kv_cache() to continue from; by default a fresh cache sized length(prompt) + n_new is created.
runtime	Optional Rfmalloc::open_fmalloc() runtime for the cache slabs (file-backed cache).

Value

A list with ids (prompt + generated, 0-based), new_ids (the generated tokens), and raw (the generated tokens decoded to bytes, or NULL when the model has no tokenizer metadata).

---

rllm_gguf_model *Load a llama-architecture GGUF model for forward passes*

Description

Reads the hyperparameters and weights of a llama-architecture 'GGUF' file into an rllm_model object usable with rllm_forward(). 2-d weight tensors are imported with Rgguf::gguf_tensor(as = "native"): their payloads keep the GGUF storage density (still q4_k/f32/... encoded) in **Rfmalloc**-backed, memory-mapped storage, and the forward pass points GGML tensors at them zero-copy. 1-d norm weights are small and are staged as packed f32 buffers.

Usage

```
rllm_gguf_model(path, runtime = NULL, rope_mode = 0L)
```

Arguments

path	Path to a GGUF file.
runtime	Optional Rfmalloc::open_fmalloc() runtime for the weight payloads; NULL uses Rfmalloc's default runtime.
rope_mode	RoPE flavour: 0 (normal/interleaved, llama) or 2 (NEOX-style, e.g. qwen2). Defaults to 0.

Details

The loader expects the standard llama tensor names (token_embd.weight, blk.<i>.attn_q.weight, ..., output_norm.weight) and hyperparameter keys (<arch>.block_count, <arch>.embedding_length, ...). Models with tied embeddings (no output.weight) reuse token_embd.weight as the output projection.

Value

An object of class rllm_model: a list with hparams (named numeric list), tensors (named list of weight payloads), and rope_mode.

See Also

rllm_forward()

---

rllm_kv_cache *Create a KV cache for incremental decoding*

Description

Allocates the per-layer key/value cache slabs an incremental rllm_forward() writes into and attends over. Each slab is a raw vector of n_ctx * (n_embd / n_head) * n_head_kv f32 values - plain R memory by default, or **Rfmalloc**-backed (file-backed, memory-mapped) when a runtime is given, which makes the cache a disk citizen: it survives in the runtime's file and its pages are evictable like any other fmalloc payload. (A quantized cache codec in the TurboQuant/PolarQuant vein can later replace the f32 slabs without touching the graph.)

Usage

```
rllm_kv_cache(model, n_ctx = 512L, runtime = NULL)
```

Arguments

model	An rllm_model from rllm_gguf_model().
n_ctx	Maximum number of positions the cache can hold.
runtime	Optional Rfmalloc::open_fmalloc() runtime for the slabs.

Details

The returned object is an environment, so `rllm_forward()` can advance its `n_past` by reference.

Value

An environment of class `rllm_kv_cache` with fields `k`, `v` (per-layer lists of raw vectors), `n_ctx`, and `n_past`.

See Also

`rllm_forward()`, `rllm_generate()`

---

rllm_quantize_tensor *Quantize a matrix into an Rfmalloc-backed quantized tensor*

Description

Encodes a dense numeric matrix into a GGUF quantized block format and stores the compressed payload in **Rfmalloc**-backed (file-backed, memory-mapped) storage, returning an `fmalloc_tensor`. This is the write-side counterpart to **Rgguf**'s `gguf_tensor(..., as = "native")`: the resulting tensor keeps its quantized storage density and, when multiplied as the right-hand operand of dense `%*% tensor` with the ggml backend active (see `rllm_use_ggml()`), is contracted natively in quantized space by GGML's SIMD-dispatched kernels without ever being decoded to double.

Usage

```
rllm_quantize_tensor(x, dtype = "q4_k", runtime = NULL)
```

Arguments

`x`	A numeric matrix to quantize.
`dtype`	Target quantized codec, one of `"q4_0"`, `"q4_1"`, `"q5_0"`, `"q5_1"`, `"q8_0"`, `"q2_k"`, `"q3_k"`, `"q4_k"` (default), `"q5_k"`, `"q6_k"`.
`runtime`	Optional **Rfmalloc** runtime handle (see `Rfmalloc::open_fmalloc()`); if `NULL`, Rfmalloc's default runtime is used.

Details

The number of rows of `x` is the quantized (per-row) dimension and must be a multiple of the codec's block size: 256 for the K-quants (`"q2_k"`, `"q3_k"`, `"q4_k"`, `"q5_k"`, `"q6_k"`) and 32 for `"q4_0"`, `"q4_1"`, `"q5_0"`, `"q5_1"`, `"q8_0"`.

Value

An `fmalloc_tensor` of the given `dtype` with `dim(x)`.

See Also

rllm_use_ggml(), Rfmalloc::create_fmalloc_tensor()

Examples

```
rt <- Rfmalloc::open_fmalloc(tempfile(fileext = ".bin"))
set.seed(1)
W <- matrix(rnorm(256 * 4, sd = 0.4), nrow = 256)  # 256 must divide nrow
Wt <- rllm_quantize_tensor(W, "q4_k", runtime = rt)
X <- matrix(rnorm(3 * 256), nrow = 3)              # 3 x 256
Y <- X %*% Wt                                      # native quantized product
dim(Y)
```

---

rllm_use_ggml *Enable or disable the ggml quantized matmul backend*

Description

Rllm registers **Rggml** as an **Rfmalloc** codec-aware matrix-multiply backend named "ggml" and selects it on load. When active, products of quantized fmalloc_tensors (types "q4_0", "q4_1", "q8_0", "q2_k", "q4_k", "q6_k") where the tensor is the right-hand operand (dense %*% tensor) are computed by ggml in quantized space, contracting each weight row through GGML's SIMD-dispatched dot kernel, with the dense operand quantized on the fly. Other products (the tensor on the left, non-quantized codecs) are declined and fall back to Rfmalloc's decode-then-BLAS path, so results are always correct regardless of the selected backend.

Usage

```
rllm_use_ggml(enable = TRUE)

rllm_backend_enabled()
```

Arguments

enable	If TRUE (default) select the "ggml" backend; if FALSE restore Rfmalloc's default BLAS path.

Details

Selection is Rfmalloc-scoped; base R's %*% is unaffected.

Value

Invisibly, TRUE if the ggml backend is active afterwards.

rllm_backend_enabled() returns TRUE if the ggml backend is the active Rfmalloc matmul backend.

See Also

rllm_quantize_tensor()

Examples

```
rllm_backend_enabled()
rllm_use_ggml(FALSE)   # fall back to Rfmalloc's BLAS decode path
rllm_use_ggml(TRUE)    # re-enable ggml quantized products
```

Index