Package: Rllm (via r-universe)

July 21, 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 borrowed typed spans, this lets a larger-than-memory model's linear layers be multiplied through 'GGML's SIMD-accelerated dot kernels zero-copy from the original model mapping. Typed semantic programs written through an R module-and-pipe architecture language describe llama, Qwen3.5, LFM2MoE, EmbeddingGemma, ESM-2, Evo 2 and Tiny Recursive Models. Bound programs join those serializable graphs to mapped parameter storage. Implemented lowerings provide persistent state, pooled embeddings, byte-level generation, and CPU or optional CUDA execution; a reference interpreter executes the same dataflow through explicit R operator tables and structured recurrence.

License GPL (>= 2)

Encoding UTF-8

Imports Rfmalloc, Rgguf, Rggml, stats

LinkingTo Rfmalloc, Rggml

Suggests tinytest

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

Config/pak/sysreqs make

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

Date/Publication 2026-07-18 21:34:09 UTC

RemoteUrl https://github.com/RGenomicsETL/Rfmalloc

RemoteRef HEAD

RemoteSha 6e05ad03613e89cab2f8bf8eb3a49d553d5a3bed

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_embed *Compute one pooled sequence embedding*

Description

Lowers an embedding model's semantic program over a complete token sequence. Bidirectional and symmetric-window attention are evaluated without a decode cache, then the program's pooling and projection pipeline produces one vector. Quantized weights remain encoded in the mapped GGUF file.

Usage

```
rllm_embed(model, tokens, normalize = TRUE, backend = c("cpu", "cuda"))
```

Arguments

model	An embedding rllm_model from rllm_gguf_model().
tokens	Integer vector of 0-based token ids.
normalize	Whether to return a unit-length vector.
backend	Compute backend. "cpu" uses mapped weights directly; "cuda" requires a CUDA-enabled Rggml installation.

Details

Tokenization is deliberately separate from model execution. Supply the model's 0-based token ids, including any BOS or EOS tokens required by its tokenizer.

Value

A numeric vector whose length is the program's output dimension.

`rllm_execute` *Execute a data-only architecture program*

Description

`rllm_execute()` interprets the dataflow and structured loops in an `rllm_program()`. Semantic operators are ordinary R functions supplied in `operators`; a small dense reference vocabulary is provided for arithmetic shared by the architecture probes. This is the dense reference interpreter for the same program that `rllm_forward()` binds to mapped weights and lowers natively through GGML.

Usage

```
rllm_execute(
  program,
  inputs,
  parameters = list(),
  counts = list(),
  operators = list()
)
```

Arguments

`program`	A data-only `rllm_program`.
`inputs`	A named list of input values. Extra entries remain available to operator functions through `context$inputs`.
`parameters`	A named list containing a value for every declared program parameter.
`counts`	A named list or atomic vector resolving symbolic loop counts and other symbolic integer attributes.
`operators`	A named list of semantic operator functions.

Details

Each operator function is called with four named arguments: `inputs`, the list of values arriving at the node; `attributes`, the node's data-only attributes; `parameters`, the named parameter values; and `context`, a list containing the current node and program, root and local inputs, symbolic counts, and the enclosing loop iterations. Operator functions control the representation of their results, so a lowering may preserve a mapped or device-backed value rather than materializing it in R.

The built-in dense reference vocabulary covers arithmetic, normalization, pooling, activations and recurrence helpers. It also covers ESM token dropout, key-padded rotary attention with attention-map results, tied projection and contact regression. Entries in `operators` replace built-ins with the same name.

Value

A named list containing every program output and tap.

`rllm_forward` *Lower a bound semantic program and return its logits*

Description

Lowers the model's inspectable `rllm_program()` to a GGML graph over its memory-mapped weights and computes it on a chosen backend. The operator vocabulary includes causal and gated attention, gated-delta recurrence, short convolution, dense gated products and sparse routed experts. Quantized weights are contracted natively in their encoded form - they are never decoded to double. The CPU backend borrows the mapped bytes directly. On its first use, the CUDA backend creates a model-owned context and uploads the codec-native weights once. Later passes reuse those resident weights; mutable inputs, cache slabs and logits move through Rggml's transfer API.

Usage

```
rllm_forward(model, tokens, cache = NULL, backend = c("cpu", "cuda"))
```

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.
`backend`	Compute backend. `"cpu"` is the zero-copy mmap path; `"cuda"` requires Rggml installed with `--with-cuda` and a visible NVIDIA device.

Details

Without a `cache`, the graph evaluates the complete token batch from zero state. With a `rllm_kv_cache()`, the pass advances every state declared by the program, including key/value, convolution and gated-delta state, then advances `cache$n_past`. This is 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,
  backend = c("cpu", "cuda")
)
```

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). A textual prompt at the start of a sequence receives the GGUF tokenizer's BOS token when one is declared. Integer prompts remain exact and are never modified.
`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).

backend — Compute backend passed to rllm_forward(). "cuda" requires a CUDA-enabled Rggml build.

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 GGUF model as a bound semantic program*

Description

Normalizes the model-family metadata into a typed rllm_program(), validates every tensor role and shape, then binds each required weight directly to the original GGUF mapping. Quantized and floating-point payloads keep their on-disk encoding; no second weight store is created. The same program is the inspectable architecture and the source consumed by the native GGML lowering.

Usage

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

Arguments

path — Path to a GGUF file.

runtime — Rfmalloc::open_fmalloc() runtime attached to the borrowed tensor views, or NULL to use the default established by Rfmalloc::init_fmalloc(). It supplies the allocation context for operations which produce fmalloc results; the weight bytes remain in the GGUF mapping.

rope_mode — Optional RoPE override: 0 for normal/interleaved or 2 for NEOX. The architecture program supplies the default.

Details

Architecture definitions are data ASTs rather than native model-family branches. Native GGML lowering covers llama, Qwen3.5, LFM2MoE and EmbeddingGemma. ESM-2 is numerically executable through rllm_execute() but its two-input program is not accepted by this native model loader. Models with tied embeddings reuse token_embd.weight as the output projection.

Value

An object of class rllm_model containing its hyperparameters, borrowed weight payloads, tokenizer metadata, and model-owned backend contexts created lazily by rllm_forward().

See Also

rllm_forward()

rllm_input *Build a typed model program with ordinary R functions and pipes*

Description

These functions form the R-facing architecture language. A program is traced from ordinary R calls, including functions returned by `rllm_module()`, residual branches and structured loops. Calling `rllm_program()` freezes the trace as data only: nodes, tensor parameters, shapes, module paths and named outputs. The resulting object contains no R closures, environments or backend pointers.

Usage

```
rllm_input(name, shape, dtype = "f32")

rllm_inputs(..., .dtype = "f32", .name = NULL)

rllm_parameter(name, shape, dtype = NULL, role = NULL)

rllm_module(name, forward)

rllm_op(x, op, ..., output_shape = NULL, output_dtype = NULL, outputs = NULL)

rllm_program(x, name = NULL)
```

Arguments

`name`	A non-empty input, parameter, module, loop or program name.
`shape`	An atomic vector describing axes. Dimensions may be positive numbers or symbolic character strings.
`dtype`	Storage or value type.
`...`	Named, data-only operator attributes, or arguments passed to a module's `forward` function.
`.dtype`	One dtype for all inputs, or a named character vector with one dtype per input.
`.name`	Program-builder name for a multiple-input trace.
`role`	Optional semantic role for a parameter.
`forward`	An ordinary R function whose first argument is a traced value. Its body may use base R's `|>` pipe.
`x`	A traced `rllm_value`, a named list of traced values for a multi-input operator, a model plan, a loaded model, or a GGUF path.
`op`	A non-empty semantic operator name.
`output_shape`	Optional result shape. The input shape is retained when omitted.
`output_dtype`	Optional result type. The input type is retained when omitted.
`outputs`	Optional named output specifications for an operator with multiple results. Each entry is either a shape or a list with `shape` and optional `dtype`. This cannot be combined with `output_shape` or `output_dtype`.

Details

This is an architecture language, not a second tensor runtime. Operators describe semantics; a backend may lower the operators it implements and must reject the rest explicitly.

Value

`rllm_input()` and single-result operators return a traced `rllm_value`. `rllm_inputs()` and multiple-result operators return named lists of traced values. `rllm_parameter()` returns a data-only tensor reference, `rllm_module()` returns a callable R function, and `rllm_program()` returns a data-only `rllm_program`.

`rllm_kv_cache` *Create program-shaped state for incremental decoding*

Description

Allocates the persistent state declared by the bound program. Attention layers receive key/value slabs, short-convolution layers receive causal history, and gated-delta layers receive both convolution history and recurrent matrices. State is plain R memory by default or **Rfmalloc**-backed when `runtime` is supplied.

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 per-layer `k`, `v`, `conv` and `recurrent` state, plus `n_ctx` and `n_past`.

See Also

`rllm_forward()`, `rllm_generate()`

rllm_linear *Typed linear, normalization, attention and pooling operators*

Description

These are convenient typed constructors over `rllm_op()`. They keep the surface close to an R module's `forward` method while recording explicit parameter identities and semantic attributes in the frozen program.

Usage

```
rllm_linear(x, weight, bias = NULL)

rllm_norm(x, weight, kind = c("rms", "layer"), eps = 1e-05, bias = NULL)

rllm_attention(
  x,
  query,
  key,
  value,
  output,
  heads,
  kv_heads = heads,
  query_norm = NULL,
  key_norm = NULL,
  rope = NULL,
  mask = list(type = "causal"),
  scale = NULL,
  state = list(op = "none")
)

rllm_pool(x, kind = c("mean", "cls", "none"))
```

Arguments

`x`	A traced `rllm_value`.
`weight`, `bias`	Tensor references from `rllm_parameter()`. `bias` is optional.
`kind`	Operator kind.
`eps`	Normalization epsilon.
`query`, `key`, `value`, `output`	Projection parameters for attention.
`heads`, `kv_heads`	Query and key/value head counts.
`query_norm`, `key_norm`	Optional per-head normalization parameters.

`rope`, `mask`, `scale`
: Data-only attention specifications.

`state`
: Data-only persistent-state specification. The default declares no state.

Value

A traced `rllm_value`.

`rllm_loop` *Trace a structured, shared recurrence*

Description

Unlike an R `for` loop, `rllm_loop()` does not unroll its body. The body is traced once into a nested data-only program and retained as a loop node. A single value supports pipe syntax. A named list supports multiple carried values, including nested recurrences with shared parameters.

Usage

```
rllm_loop(x, times, body, name = "loop")
```

Arguments

`x`
: A traced value or named list of values from one program.

`times`
: A positive integer or one symbolic count.

`body`
: A function returning the same value structure as `x`.

`name`
: Loop name.

Value

A traced value, or a named list of traced values when `x` is a list.

`rllm_plan` *Inspect the normalized checkpoint description of a GGUF model*

Description

The executable `rllm_program()` is the source of truth. `rllm_plan()` derives a compact layer-oriented inspection view from the program's validated GGML lowering. Constructing or loading a model does not retain this second representation.

Usage

```
rllm_plan(x)
```

Arguments

x	An rllm_model or a path to a GGUF file.

Value

An inspectable object of class rllm_plan.

---

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_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"; "q2_0" uses GGML's group-64 ternary blocks.

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_residual *Trace a residual branch*

Description

`branch` is evaluated immediately with `x`. Its nodes are recorded, then an explicit add node joins the original value and the branch result.

Usage

```
rllm_residual(x, branch)
```

Arguments

`x`	A traced `rllm_value`.
`branch`	A function of one traced value.

Value

A traced `rllm_value`.

---

rllm_tap *Name an intermediate program output*

Description

Name an intermediate program output

Usage

```
rllm_tap(x, name)
```

Arguments

`x`	A traced `rllm_value`.
`name`	A unique output name.

Value

x, invisibly annotated in its builder so a pipe may continue.

`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_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