Package: relm (via r-universe)

July 9, 2026

Title Local Large Language Models as Base-R Objects

Version 0.1.0

Description Load and run local large language models (through a vendored, patched 'llama.cpp') from R and work with them as plain data frames and matrices in base-R idiom. Provides model loading and tokenization ('llm', 'llm_tokens'), text generation ('llm_generate'), next-token distributions ('llm_logits'), text embeddings ('llm_embed'), and mechanistic interpretability tools -- activation tracing ('llm_trace'), activation steering ('llm_steer'), and neuron ablation ('llm_ablate'). A checksum-verified downloader ('llm_download') fetches pinned models.

License MIT + file LICENSE | Apache License (== 2.0)

URL https://github.com/Vadale/R-ebirth

BugReports https://github.com/Vadale/R-ebirth/issues

Depends R (>= 4.5.0)

Imports nanoarrow

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Suggests testthat (>= 3.0.0), glmnet, uwot, dbscan, quarto

VignetteBuilder quarto

Config/testthat/edition 3

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.71.0, cmake (>= 3.28), xz

Config/pak/sysreqs cmake libzstd-dev xz-utils libclang-dev

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

Date/Publication 2026-07-09 10:58:52 UTC

RemoteUrl https://github.com/Vadale/R-ebirth

RemoteRef HEAD

RemoteSha a50502b3966f9429c662395c68268650a5ced104

RemoteSubdir rebirth

Contents

as.matrix.relm_trace . . . 2
close.llm . . . 3
llm . . . 4
llm_ablate . . . 5
llm_download . . . 7
llm_embed . . . 8
llm_generate . . . 10
llm_logits . . . 11
llm_steer . . . 12
llm_tokens . . . 14
llm_trace . . . 15
summary.llm . . . 18

Index **19**

`as.matrix.relm_trace` *Extract one activation slice as a matrix*

Description

Pulls a single `(layer, component)` slice out of a `llm_trace()` result as a base numeric `matrix`: one row per captured `(prompt_id, token_pos)` and one column per neuron (`hidden_size` wide). Row names are `"<prompt_id>.<token_pos>"`. This is the bridge from the long-format trace to matrix tools such as `stats::prcomp()`.

Usage

```
## S3 method for class 'relm_trace'
as.matrix(x, layer, component = "residual", ...)
```

Arguments

`x`	A `relm_trace` from `llm_trace()`.
`layer`	Required single 1-based layer index; it must be present in the trace.
`component`	Single component name (default `"residual"`); it must be present in the trace.
`...`	Ignored.

Value

A numeric `matrix`, rows = captured `(prompt_id, token_pos)` (row names `"<prompt_id>.<token_pos>"`), columns = neurons.

See Also

`llm_trace()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
tr <- llm_trace(m, "The cat sat.", layers = 1:2)
as.matrix(tr, layer = 1, component = "residual")
close(m)
```

---

close.llm *Free a model handle*

Description

Deterministically frees the native memory behind an `llm` handle. On a memory-constrained machine this lets you release several gigabytes immediately rather than waiting for garbage collection (the finalizer remains the safety net). A double close is a no-op; any later use of the handle raises `relm_error_closed`.

Usage

```
## S3 method for class 'llm'
close(con, ...)
```

Arguments

`con`	An `llm` handle.
`...`	Ignored (present for compatibility with the `close()` generic).

Value

`invisible(NULL)`.

See Also

`llm()`

`llm` *Load a local large language model*

Description

Loads a GGUF model file and returns an `llm` handle: an external pointer to the native model plus its metadata. All arguments are validated in R before the native boundary is crossed, so a bad request is reported as a classed condition and never reaches the engine.

Usage

```
llm(
  path,
  context_length = 4096,
  gpu_layers = NULL,
  backend = c("auto", "metal", "cuda", "cpu"),
  mmap = TRUE
)

## S3 method for class 'llm'
print(x, ...)
```

Arguments

`path`	Single string: path to a GGUF model file.
`context_length`	Positive integer: the active context window in tokens (llama.cpp: `n_ctx`). Default 4096.
`gpu_layers`	`NULL` (auto: offload every layer that fits) or a single non-negative integer count of layers to offload (llama.cpp: `n_gpu_layers`). Ignored on the CPU backend.
`backend`	One of `"auto"`, `"metal"`, `"cuda"`, `"cpu"`.
`mmap`	Logical: memory-map the model file (default `TRUE`).
`x`	An `llm` handle.
`...`	Ignored.

Details

The handle owns native memory (potentially several gigabytes). Free it deterministically with close() when done; a garbage-collection finalizer frees it as a safety net otherwise (see `close.llm()`).

`backend = "auto"` resolves to the fastest backend this build supports (Metal on Apple silicon, otherwise CPU). Requesting a backend the build was not compiled with raises `relm_error_backend`.

No model ships inside the package yet, so the runnable example is guarded by the `RELM_TEST_MODEL_QWEN` environment variable (point it at a local Qwen2.5 GGUF to run it). A tiny in-repo model arrives in a later work package.

Value

An object of class `llm` (see the package's class documentation).

`x`, invisibly.

See Also

`close.llm()`, `print.llm()`, `summary.llm()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
print(m)
summary(m)
close(m)
```

`llm_ablate` *Ablate neurons in a model*

Description

Returns a **new** `llm` handle with the listed 1-based `neurons` of `component` at `layer` forced to `value` during every subsequent forward pass (`llm_generate()`, `llm_logits()`). The original handle is untouched: to remove the ablation, use the original object (reversibility is exact, D-003).

Usage

```
llm_ablate(m, layer, neurons, value = 0, component = "residual")
```

Arguments

`m`	An `llm` handle from `llm()`.
`layer`	Single 1-based transformer block, in `1:m$layers`.
`neurons`	A non-empty vector of 1-based neuron indices in `1:m$hidden_size` to force to `value`.
`value`	Single finite number the neurons are forced to (default `0`).
`component`	Which sub-layer to ablate. Only `"residual"` (the default) is supported in this release.

Details

Ablation forces `x[neuron] := value` at the residual choke point – a native graph op (`x * mask + add`), reversible by construction. Unlike steering it covers **every** layer, including layer 1.

Composition. Deriving from an already-steered or ablated handle carries its full spec forward and adds the new one, then builds a single fresh context from the original weights (contexts are never chained). Ablation is a **union, last-write-wins** per `(layer, neuron)` – a later ablation of the same neuron overrides its value. The semantics are **derivation-order-independent**: `m |> llm_ablate(...) |> llm_steer(...)` produces the same forward pass as `m |> llm_steer(...) |> llm_ablate(...)`, and a steer never moves an ablated neuron (the ablation runs after the steer at the graph, `(x + steer) * mask + add`, forcing the neuron to `value`, D-016).

Not free. Each `llm_ablate()`/`llm_steer()` call allocates a fresh context on the shared weights – a sub-second pause and real memory (its own KV cache), not a cheap copy.

Generation/logits only (for now). Interventions apply to `llm_generate()` and `llm_logits()`. `llm_embed()` and `llm_trace()` on an intervened handle raise `relm_error_embed` / `relm_error_trace` rather than silently returning base vectors mislabeled as ablated.

Only `component = "residual"` ablation is supported in this release (the shared choke point); `"attn_out"`/`"mlp_out"` raise `relm_error_intervention`.

Model support. Interventions work on any standard-residual decoder; before the handle is returned, a runtime probe verifies on *this* model that the ablation takes effect at each requested layer, raising `relm_error_intervention` rather than silently doing nothing. `llama` and `qwen2` are additionally *behaviorally validated* (the valence / KL acceptance fixtures); other architectures are enabled once they pass the probe.

Value

A new `llm` handle with the ablation added to its accumulated `interventions`; the source handle is returned unchanged. `print()` shows the active intervention count and `summary()` lists them.

See Also

`llm_steer()`, `llm()`, `llm_generate()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
ablated <- llm_ablate(m, layer = 5, neurons = c(10, 42, 128))
# Composition is order-independent: these two derivations behave identically.
dir <- rep(0.05, m$hidden_size)
a <- m |> llm_ablate(layer = 5, neurons = 10) |> llm_steer(layer = 6, direction = dir)
b <- m |> llm_steer(layer = 6, direction = dir) |> llm_ablate(layer = 5, neurons = 10)
summary(a)
close(a)
close(b)
close(ablated)
close(m)
```

`llm_download` *Download a pinned model, verified by checksum*

Description

Fetches a GGUF model file over HTTPS into a local cache and returns its path. `model` is either a pinned alias from the package's model registry (see the Details section) or a full `https://` URL. The download is **fail-closed**: for a registry alias the file's SHA256 must match the pinned value or the file is deleted and `relm_error_download` is raised. Nothing downloaded is ever executed — the file is only written to disk and checksummed.

Usage

```
llm_download(model, dir = NULL, quiet = FALSE)
```

Arguments

`model`	Single string: a registry alias (see Details) or a full `https://` URL to a GGUF file.
`dir`	`NULL` (the user cache directory) or a single string naming the directory to download into (created recursively if missing). Use a directory only you can write to; downloading into a world-writable/shared directory is not supported.
`quiet`	Single logical. `TRUE` suppresses the download progress bar and the informational messages. Default `FALSE`.

Details

Registry aliases. The package ships a small registry (`inst/models.csv`) of license-clean, checksum-pinned models. The current aliases are:

- `"qwen2.5-0.5b-instruct-q8_0"` — Qwen2.5 0.5B Instruct, Q8_0 (Apache-2.0); the small CI-integration model.
- `"qwen2.5-1.5b-instruct-q4_k_m"` — Qwen2.5 1.5B Instruct, Q4_K_M (Apache-2.0); the demo default.

Passing an unknown alias raises `relm_error_download` and lists the known aliases. Larger Qwen quantizations (7B) ship as split multi-part GGUFs and are not in the single-file registry yet; Gemma models are gated by the Gemma Terms of Use (they require accepting terms and an access token on Hugging Face), so they are supplied by local path rather than downloaded here.

Full URLs. A full `https://` URL is downloaded as given. Only HTTPS is accepted (`http://`, `ftp://`, `file://`, ... are rejected). A bare URL has no pinned checksum, so the file cannot be *verified*; it is downloaded and its computed SHA256 is reported so you can pin it — the function never presents an unverifiable file as if it had been verified.

Caching. `dir = NULL` uses the per-user cache directory `tools::R_user_dir("relm", "cache")`; the directory is created if missing. The target file name is the URL's last path segment. If a registry

model is already present and its checksum matches, the download is skipped (idempotent, offline-friendly); a present-but-mismatching file is treated as corrupt and re-downloaded. A bare URL has no pinned checksum, so a same-named cached file cannot be shown to have come from that URL and is never reused – a bare URL is always re-fetched. Verification happens on a temporary file that is only moved into place once it passes, so the final path never holds unverified bytes. Downloads follow HTTPS redirects; for a registry alias the pinned checksum still guards integrity end-to-end, so a redirect cannot substitute the bytes.

This is one of only two functions in the package that write outside a session temporary directory (the other is `llm_trace()` spill); see the package's side-effect contract.

Value

The local file path, returned **invisibly**. Errors: `relm_error_argument` (bad `model`/`dir`/`quiet` type), `relm_error_download` (non-HTTPS URL, unknown alias, network failure, checksum mismatch, or an unwritable directory).

See Also

`llm()`

Examples

```
# The default download directory (dir = NULL); no network access is performed:
tools::R_user_dir("relm", "cache")

## Not run:
# Fetch a pinned, checksum-verified model and load it:
path <- llm_download("qwen2.5-0.5b-instruct-q8_0")
m <- llm(path)

## End(Not run)
```

`llm_embed` *Embed text with a model*

Description

Encodes each string in `x` into a fixed-length numeric vector, returning a base `matrix` with one row per input and one column per embedding dimension (so the matrix is `length(x)` by the model's `hidden_size`).

Usage

```
llm_embed(m, x, pooling = c("mean", "last", "model"), normalize = TRUE)
```

Arguments

m	An llm handle from llm().
x	A character vector of one or more non-empty strings to embed; NA and empty strings ("") are rejected. names(x) become the row names.
pooling	How to reduce each input's per-token vectors to one vector: "mean" (average), "last" (final token), or "model" (the model's own pooling when the GGUF defines one; otherwise an error asking for "mean"/"last").
normalize	Single logical. TRUE (default) L2-normalizes each row so rows are unit vectors and dot products are cosine similarities.

Details

Each input is tokenized (with the model's begin/end-of-sequence markers) and run through a forward pass in a dedicated embeddings-mode context; the per-token hidden states are then pooled into one vector per input by pooling. "mean" averages the token vectors, "last" takes the final token's vector, and "model" uses the model's own pooling when the GGUF defines one (a purely generative model such as Qwen2.5 defines none, so "model" raises relm_error_embed asking for "mean" or "last").

With normalize = TRUE (the default) each row is L2-normalized to a unit vector, so the dot product of two rows is their cosine similarity. A zero vector is returned unchanged (never NaN). normalize is validated and its effect is explicit — there is no silent normalization.

The model must carry a tokenizer; a no_vocab model raises relm_error_tokenize. No model ships inside the package yet (the in-repo synthetic model has no tokenizer), so the runnable example is guarded by the RELM_TEST_MODEL_QWEN environment variable — point it at a local Qwen2.5 GGUF to run it.

Embedding an **intervened** handle (from llm_steer()/llm_ablate()) raises relm_error_embed: interventions currently apply to generation and logits only, and the embedding context does not inherit them, so returning base vectors labeled as intervened would be silent mislabeling. Embed the original handle.

Value

A numeric matrix, length(x) rows by the model's embedding size (columns), with row names names(x) when set, else the input positions as characters.

See Also

llm(), llm_tokens(), llm_generate()

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
e <- llm_embed(m, c(a = "cats and dogs", b = "domestic pets"))
dim(e)
close(m)
```

`llm_generate` *Generate text from a model*

Description

Autoregressively continues each `prompt`. With `chat = TRUE` (the default) the prompt is wrapped as a user turn using the model's own chat template, so the formatting matches what the model was trained on; with `chat = FALSE` the prompt is completed verbatim.

Usage

```
llm_generate(
  m,
  prompt,
  max_tokens = 256,
  temperature = 0.8,
  top_p = 0.95,
  seed = NULL,
  chat = TRUE,
  stop = NULL
)
```

Arguments

`m`	An `llm` handle from `llm()`.
`prompt`	A character vector of prompts; the result has one element per prompt and preserves `names(prompt)`.
`max_tokens`	Single positive integer: the maximum number of tokens to generate per prompt.
`temperature`	Single non-negative number. `0` is greedy (deterministic); higher values sample more diversely.
`top_p`	Single number in `(0, 1]`: nucleus sampling keeps the most probable tokens whose cumulative probability reaches `top_p`.
`seed`	`NULL` (draw and record a seed) or a single non-negative whole number for a reproducible run.
`chat`	Single logical. `TRUE` applies the model's chat template; `FALSE` completes the raw prompt. If the model's embedded template cannot be detected by the engine (e.g. some Gemma models), a built-in template for the model's architecture is used instead; if neither applies, a classed error is raised rather than mis-formatting the prompt.
`stop`	`NULL`, or a character vector of stop sequences that end generation.

Details

Decoding is greedy when `temperature = 0` — the exact, reproducible path — and temperature + nucleus (top-p) sampling otherwise. Sampling is drawn on the CPU from a seeded generator, so a run is fully reproducible: the same `seed` and arguments produce the same text across runs and sessions. When `seed = NULL` a seed is drawn (from R's RNG, so `set.seed()` makes even that reproducible) and **recorded** — the seed actually used is always returned as `attr(result, "seed")`, so any generation can be replayed.

Generation stops at `max_tokens`, at the model's end-of-generation token, or as soon as one of the `stop` strings appears (the output is truncated just before it). A prompt longer than the model's context window raises `relm_error_context_overflow`, whose message states by how much.

Value

A character vector the same length as `prompt` (names preserved), each element the generated continuation. The seed used is attached as `attr(result, "seed")`.

See Also

`llm()`, `llm_tokens()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
llm_generate(m, "In one sentence, what is R?", max_tokens = 40, seed = 1)
close(m)
```

`llm_logits` *Inspect the next-token distribution*

Description

Runs a forward pass over each `prompt` and returns the model's distribution for the single token that would come next: the `top` most likely candidates, ranked from most to least probable. This is the raw material behind generation — the first token `llm_generate()` would pick (greedily) is exactly this table's rank-1 token.

Usage

```
llm_logits(m, prompt, top = 20)
```

Arguments

`m`	An `llm` handle from `llm()`.
`prompt`	A character vector of prompts; the result has `top` rows per prompt, and `prompt_id` is the 1-based index into `prompt`.
`top`	Single positive integer: how many top candidates to return per prompt (clamped to the vocabulary size). Default 20.

Details

Each prompt is treated as raw text (no chat template is applied — wrap it yourself if you want the chat framing) and tokenized the way `chat = FALSE` generation is. The distribution is read at the final position, so it answers "what token does the model expect immediately after this text?". Probabilities are the softmax over the **whole** vocabulary, computed before the `top` candidates are selected, so each `prob` is the token's true share of the full distribution: the returned probabilities are correct but (being only the head of the distribution) sum to less than 1.

The result is one long-format `data.frame` with `top` rows per prompt, stacked in prompt order. Within each prompt the rows are ordered by descending logit, so `rank == 1` is the most likely next token; `logit` and `prob` are therefore non-increasing down the ranks. Token ids are **1-based** (like every index in `relm`, and like `llm_tokens()`), so `token_id` round-trips through `llm_tokens(m, token_id, decode = TRUE)`.

Active interventions on `m` (from `llm_steer()`/`llm_ablate()`) apply, so `llm_logits()` is the direct way to read how a steer or an ablation reshapes the next-token distribution.

The model must carry a tokenizer; a `no_vocab` model (such as the in-repo synthetic model) raises `relm_error_tokenize`. No model ships inside the package yet, so the runnable example is guarded by the `RELM_TEST_MODEL_QWEN` environment variable — point it at a local Qwen2.5 GGUF to run it.

Value

A base `data.frame` with `top` rows per prompt and columns `prompt_id` (int, 1-based index into `prompt`), `rank` (int, 1 = most likely), `token_id` (int, 1-based), `token` (chr, the token piece), `logit` (dbl), and `prob` (dbl, softmax over the full vocabulary).

See Also

`llm()`, `llm_generate()`, `llm_tokens()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
llm_logits(m, "The capital of France is", top = 5)
close(m)
```

`llm_steer` *Steer a model along a direction*

Description

Returns a **new** `llm` handle that adds `coef * direction` to the residual stream at `layer` during every subsequent forward pass (`llm_generate()`, `llm_logits()`). The original handle is untouched: to remove the steering, use the original object (reversibility is exact, D-003). Interventions compose – see Details.

Usage

```
llm_steer(m, layer, direction, coef = 1, positions = "all")
```

Arguments

`m`	An `llm` handle from `llm()`.
`layer`	Single 1-based transformer block to steer, in `2:m$layers` (layer 1 is not steerable – see Details).
`direction`	Numeric vector of length `m$hidden_size`: the steering direction in residual space (finite, no `NA`).
`coef`	Single finite number scaling `direction` (default `1`). Negative values steer the opposite way.
`positions`	Which token positions to steer. Only `"all"` (the default) is supported in this release.

Details

Steering is llama.cpp's native control-vector mechanism: a per-layer, `hidden_size`-wide vector added to the residual at `layer`'s output, for **all token positions** (hence `positions = "all"` is the only supported value in this release; a position subset raises `relm_error_intervention`).

Composition. Deriving from an already-steered or ablated handle carries its full spec forward and adds the new one, then builds a single fresh context from the original weights (contexts are never chained). Steering **stacks by summation** – two steers on the same layer add – and the semantics are **derivation-order-independent**: `m |> llm_steer(...) |> llm_ablate(...)` produces the same forward pass as `m |> llm_ablate(...) |> llm_steer(...)`. A steer never moves an ablated neuron: at the graph the ablation runs after the steer (`(x + steer) * mask + add`), so a jointly steered-and-ablated neuron is forced to the ablation `value` (D-016).

Layer 1 is not steerable. The native control vector reserves engine index 0 and has no slot for the first transformer block, so `layer = 1` raises `relm_error_intervention`. Steer a later layer (`2:m$layers`), or ablate layer 1 with `llm_ablate()` (ablation covers every layer).

Not free. Each `llm_steer()`/`llm_ablate()` call allocates a fresh context on the shared weights – a sub-second pause and real memory (its own KV cache, ~hundreds of MB on the demo models), not a cheap copy. Hold a handful of intervened handles, not hundreds.

Generation/logits only (for now). Interventions apply to `llm_generate()` and `llm_logits()`. `llm_embed()` and `llm_trace()` build their own fresh contexts that do not inherit the adapters, so calling them on an intervened handle raises `relm_error_embed` / `relm_error_trace` rather than silently returning base (un-intervened) vectors mislabeled as steered.

Model support. Interventions work on any standard-residual decoder. Before the handle is returned, a runtime probe verifies on *this* model that steering actually shifts the residual at each requested layer; if it would silently do nothing (an architecture that does not route its residual through the choke point the mechanism hooks), `relm_error_intervention` is raised instead of a no-op handle. The `llama` and `qwen2` architectures are additionally *behaviorally validated* – they pass the valence-steering and KL-ablation acceptance fixtures; any other architecture is enabled the moment it passes the probe.

Value

A new `llm` handle with the steering added to its accumulated `interventions`; the source handle is returned unchanged. `print()` shows the active intervention count and `summary()` lists them.

See Also

`llm_ablate()`, `llm()`, `llm_generate()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
# A steering direction is any hidden_size-wide vector (here a placeholder).
dir <- rep(0.05, m$hidden_size)
steered <- llm_steer(m, layer = 8, direction = dir, coef = 2)
summary(steered) # lists the active intervention
# The original handle is untouched -- generation reproduces the base output.
identical(
  llm_generate(m, "Tell me about the sea.", max_tokens = 20, seed = 1),
  llm_generate(m, "Tell me about the sea.", max_tokens = 20, seed = 1)
)
close(steered)
close(m)
```

`llm_tokens` *Tokenize text, or decode token ids back to text*

Description

Converts between text and the model's tokens. With `decode = FALSE` (the default) `x` is text and the result is the token ids; with `decode = TRUE` `x` is token ids and the result is the reconstructed string.

Usage

```
llm_tokens(m, x, decode = FALSE)
```

Arguments

`m`	An `llm` handle.
`x`	For encoding, a character vector; for decoding, an integer vector of 1-based token ids.
`decode`	Single logical. `FALSE` (default) encodes text to ids; `TRUE` decodes ids to text.

Details

Encoding (`decode = FALSE`). `x` is a character vector. Each element is tokenized into a **named integer vector**: the values are the token ids and the names are the token pieces (the text each id renders as). For a single string a named integer vector is returned; for several strings a list of such vectors is returned, preserving `names(x)`. No beginning/end-of-sequence markers are added (you get exactly the tokens of the text); chat formatting is `llm_generate`'s job, not this function's.

Decoding (`decode = TRUE`). `x` is an integer vector of token ids and the result is a single string. Decoding is UTF-8 correct even when a multi-byte character spans two tokens — always decode the whole id vector rather than concatenating the piece names, which can split a character.

Token ids are **1-based** in the R API (like every other index in `relm`); subtract 1 to compare them with a raw vocabulary index (llama.cpp / Hugging Face). Encoding and decoding are exact inverses: `llm_tokens(m, llm_tokens(m, txt), decode = TRUE)` reproduces `txt`.

The model must carry a tokenizer; a `no_vocab` model raises `relm_error_tokenize`.

Value

Encoding: a named integer vector (single input) or a list of them (several inputs). Decoding: a single string.

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
ids <- llm_tokens(m, "The quick brown fox")
ids
llm_tokens(m, ids, decode = TRUE)
close(m)
```

---

llm_trace *Trace a model's activations*

Description

Runs a forward pass over each prompt's tokens (no sampling) and captures the internal activations selected by the filters, returning a long-format `relm_trace` `data.frame` with one row per `(prompt, token position, layer, component, neuron)`.

Usage

```
llm_trace(
  m,
  prompts,
  layers = NULL,
  positions = "last",
  components = "residual",
  spill = TRUE,
```

```
  spill_dir = NULL
)

## S3 method for class 'relm_trace'
print(x, ...)

## S3 method for class 'relm_trace'
summary(object, ...)

## S3 method for class 'summary.relm_trace'
print(x, ...)
```

Arguments

m	An llm handle from llm().
prompts	A character vector of one or more non-empty prompts; NA and empty strings ("") are rejected. names(prompts) are retained on the prompts attribute.
layers	NULL (default: every block) or a vector of 1-based block indices to capture (each in 1:m$layers).
positions	Which token positions to capture: "last" (default, the last token of each prompt), "all", or a vector of 1-based positions.
components	A subset of c("residual", "attn_out", "mlp_out") (default "residual"): the residual stream, the attention sub-layer output (after the output projection; TransformerLens hook_attn_out), and/or the MLP sub-layer output. Tracing is supported on the llama, qwen2, gemma3, qwen3, qwen35, and gemma4 architectures; any other raises relm_error_trace. "residual" is available on all of them. "mlp_out" is available on all except gemma4 (whose mixture-of-experts layers name their FFN output differently, so capturing it would silently miss layers). "attn_out" is currently observable only on llama-family models; on the others (which name only the pre-projection attention tensor, a different quantity) requesting it raises relm_error_trace listing the available components rather than silently substituting a different tensor.
spill	Single logical (default TRUE). When a capture exceeds the memory budget, TRUE streams it to a disk file (a lazily-loaded spilled trace) and FALSE raises relm_error_oom instead. A within-budget capture is always held in memory regardless of this flag.
spill_dir	NULL (default: a managed per-session directory under the user cache, cleaned up when the session ends) or a single directory path in which to write spill files (left in place for you to manage).
x	A relm_trace (for print/as.matrix) or its summary.
...	Ignored.
object	A relm_trace.

Details

llm_trace() is the anatomy lab's core tool: it observes the residual stream and the attention/MLP sub-layer outputs as the model processes text, so those activations can be analysed with ordinary R

(PCA, per-layer probes, and so on). The tap adds no overhead to normal generation — it runs on a dedicated, transient context created only for the trace.

The captured columns are exactly (in order): `prompt_id` (1-based index into `prompts`), `token_pos` (1-based position within that prompt), `token` (the token piece), `layer` (1-based transformer block), `component` (`"residual"`, `"attn_out"`, or `"mlp_out"`), `neuron` (1-based index within the component vector), and `value` (the activation).

Memory (the 16 GB rule). A full trace can be large, so the defaults capture little (`positions = "last"`, `components = "residual"`); widen them deliberately. Before running, the size of the `data.frame` you would receive is estimated from the filters (the materialized-object cost, D-017). If it fits the budget (`min(2 GB, 20% of RAM)`, overridable with `options(relm.trace_budget = <bytes>)`) it is held in memory. If it exceeds the budget and `spill = TRUE` (the default), the capture is streamed to an Arrow-IPC file under the session cache and the result is a *spilled* `relm_trace` that loads lazily: `print()`/`summary()` never read the file, and `as.matrix.relm_trace()` reads only the requested `(layer, component)` slice. If it exceeds the budget and `spill = FALSE`, the call raises `relm_error_oom` *before* any allocation, its `estimate_bytes` field stating the estimate. Spill files are removed when the R session ends.

The model must carry a tokenizer; a `no_vocab` model raises `relm_error_tokenize`. As with the other text entry points, the runnable example is guarded by the `RELM_TEST_MODEL_QWEN` environment variable.

Tracing an **intervened** handle (from `llm_steer()`/`llm_ablate()`) raises `relm_error_trace`: interventions currently apply to generation and logits only, and the trace context does not inherit them, so tracing would capture the base (un-intervened) forward pass while labeling it intervened. Trace the original handle.

Value

A `relm_trace`: a `data.frame` (class `c("relm_trace", "data.frame")`) with the seven columns above, carrying `model`, `spilled`, `spill_files`, and `prompts` attributes. When `spilled` is `TRUE` the rows live in `spill_files` (read on demand by `as.matrix.relm_trace()`) rather than in the frame. See `as.matrix.relm_trace()` to extract one `(layer, component)` slice as a numeric matrix.

`print` returns its argument invisibly.

`summary` returns a `data.frame` (class `summary.relm_trace`) with one row per captured `(layer, component)` group: its `n` and mean `|value|`. For a spilled trace, `mean_abs` is `NA` (reporting it would force a data load); use `as.matrix.relm_trace()` to read a slice.

See Also

`llm()`, `llm_embed()`, `as.matrix.relm_trace()`

Examples

```
m <- llm(Sys.getenv("RELM_TEST_MODEL_QWEN"))
tr <- llm_trace(m, c("The cat sat.", "Quarks bind."), layers = 1:4)
tr
summary(tr)
x <- as.matrix(tr, layer = 1, component = "residual")
dim(x)
close(m)
```

summary.llm *Summarize a model handle*

Description

Returns a classed list with the print-level metadata plus the model's memory footprint, tokenizer (vocabulary) information, and the full list of active interventions. Its own `print` method renders it.

Usage

```
## S3 method for class 'llm'
summary(object, ...)

## S3 method for class 'summary.llm'
print(x, ...)
```

Arguments

`object`	An `llm` handle.
`...`	Ignored.
`x`	A `summary.llm` object.

Value

An object of class `summary.llm`.

`x`, invisibly.

Index

as.matrix.relm_trace, 2
as.matrix.relm_trace(), *17*

close(), *4*
close.llm, 3
close.llm(), *4, 5*

llm, 4
llm(), *3*, *5, 6*, *8–14*, *16, 17*
llm_ablate, 5
llm_ablate(), *9*, *12–14*, *17*
llm_download, 7
llm_embed, 8
llm_embed(), *6*, *13*, *17*
llm_generate, 10
llm_generate(), *6*, *9*, *11, 12*, *14*
llm_logits, 11
llm_steer, 12
llm_steer(), *6*, *9*, *12*, *17*
llm_tokens, 14
llm_tokens(), *9*, *11, 12*
llm_trace, 15
llm_trace(), *2, 3*, *6*, *8*, *13*

print.llm *(*llm*)*, 4
print.llm(), *5*
print.relm_trace *(*llm_trace*)*, 15
print.summary.llm *(*summary.llm*)*, 18
print.summary.relm_trace *(*llm_trace*)*, 15

stats::prcomp(), *2*
summary.llm, 18
summary.llm(), *5*
summary.relm_trace *(*llm_trace*)*, 15