Package: relm (via r-universe)

October 29, 2026

Title Local Large Language Models as Base-R Objects

Version 0.5.0

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()), embeddings (llm_embed()), and mechanistic interpretability tools -- activation tracing (llm_trace()), activation steering (llm_steer()), neuron ablation (llm_ablate()), and grouped statistical probes (llm_probe()). Vision-language models loaded with their projector take image input in generation and embeddings. A checksum-verified downloader (llm_download()) fetches pinned models.

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

Copyright See inst/AUTHORS, inst/RUST-NOTICES and inst/THIRD-PARTY-NOTICES for bundled source authorship, copyrights and licenses.

URL https://github.com/Vadale/R-ebirth, https://vadale.r-universe.dev/relm

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

Depends R (>= 4.5.0)

Imports methods, nanoarrow

Encoding UTF-8

RoxygenNote 8.0.0

Suggests testthat (>= 3.0.0), later (>= 1.4.8), promises (>= 1.5.0), glmnet, uwot, dbscan, quarto

VignetteBuilder quarto

Config/testthat/edition 3

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager) >= 1.85.0, rustc >= 1.85.0, cmake (>= 3.28), xz

NeedsCompilation yes

Author Alessandro Vadala [aut, cre, cph], ggml authors [ctb, cph] (Bundled llama.cpp and ggml (MIT)), Authors of the bundled Rust and native dependencies [ctb, cph] (Individual authors, copyrights and licenses are listed in AUTHORS, RUST-NOTICES and THIRD-PARTY-NOTICES)

Maintainer Alessandro Vadala <vadale93@gmail.com>

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

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

Date/Publication 2026-10-29 21:15:12 UTC

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

RemoteRef HEAD

RemoteSha 6f07fe68877dcbd0d75ed1587028fb2e93a47de0

Contents

activations . . . 3
as.matrix.relm_trace . . . 3
close.llm . . . 4
llm . . . 5
llm_ablate . . . 7
llm_apply_direction . . . 8
llm_cancel . . . 9
llm_compare . . . 10
llm_direction . . . 11
llm_download . . . 13
llm_embed . . . 14
llm_generate . . . 17
llm_logits . . . 23
llm_probe . . . 25
llm_steer . . . 26
llm_tokens . . . 28
llm_trace . . . 29
plot.llm . . . 32
plot.relm_timeline . . . 33
predict.llm_probe . . . 34
probe-methods . . . 35
summary.llm . . . 36

Index **37**

activations *Select activation features inside a probe formula*

Description

A formula marker interpreted by `llm_probe()`. Calling it directly is an error.

Usage

```
activations(layer, component = "residual")
```

Arguments

`layer`	Distinct positive 1-based layer indices.
`component`	One of `"residual"`, `"attn_out"` or `"mlp_out"`.

Value

No standalone value; identifies activation slices inside a probe formula.

Examples

```
label ~ activations(layer = 1:3, component = "residual")
```

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

Frees the native memory behind an llm handle when native execution is idle. 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. While async generation owns the native execution domain, closing any handle marks it closed immediately and defers native destruction until safe. Closing the submitting handle also requests cancellation. A pending promise keeps its submitting handle alive when ordinary R references are removed.

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,
  projector = NULL
)

## 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`).
`projector`	`NULL` (default: text-only, unchanged) or a single string: path to the model's **mmproj GGUF**, enabling image input (see the *Image input* section).
`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"` selects available compiled backends in priority order: Metal, then CUDA, then CPU. It does not benchmark their speed. 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.

Image input (vision models)

projector enables image input for a vision-language model: point it at the model's companion **mmproj GGUF** (the vision projector published alongside the model, e.g. mmproj-*.gguf). The projector is a session property fixed at load — it is bound to the loaded model — so it belongs on llm(), not on each call; a handle loaded with a projector then accepts the images argument of both llm_generate() and llm_embed(). A projector whose input embedding size does not match the model raises relm_error_image naming both sizes.

Trust note: the projector file is engine-trusted input, exactly like path — a corrupt or hostile GGUF is parsed by native code. Prefer files whose SHA256 you can verify (e.g. fetched with llm_download()); treat a projector from an unknown source with the same care as a model file.

See Also

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

Examples

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


# A vision-language model: pass its companion mmproj as the projector
# (e.g. llm_download("qwen2-vl-2b-instruct-q4_k_m") and
# llm_download("qwen2-vl-2b-instruct-mmproj-f16")).
m <- llm(Sys.getenv("RELM_TEST_MODEL_VLM"),
  projector = Sys.getenv("RELM_TEST_MMPROJ_VLM")
)
print(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_apply_direction` *Apply a compatible recorded direction*

Description

Validate a trusted direction artifact, then derive an additive residual or static component-projection handle using native capability checks.

Usage

```
llm_apply_direction(
  m,
  direction,
  context,
  coef = 1,
  max_bytes = 64 * 1024^2,
  operator = "add"
)
```

Arguments

`m`	A live `llm` handle. Its metadata is checked for compatibility, not treated as authenticated weight identity.
`direction`	A `relm_direction` returned by `llm_direction()`.

`context`	For construction, a recorded list with `model`, `capture`, `pairs`, `splits` and `seed`; see Details and the contrast-directions vignette. For application, an independently recorded destination `model` record.
`coef`	One finite coefficient. For projection, zero is an exact no-op, one removes the direction's component, and negative values amplify it.
`max_bytes`	Whole materialization budget, 1 MiB through 512 MiB; default 64 MiB. Includes input matrices and metadata, working arrays and output.
`operator`	Exactly `"add"` (default) for schema-1 residual directions or `"project"` for schema-2 MLP/attention directions; no partial matching.

Details

The independently recorded destination context must match all six fields of the construction model record exactly. Verify the model file and keep it unchanged while loading and experimenting. Same path or width alone is insufficient. This check cannot authenticate already-loaded weights. Projection edits each executed component row as `h - coef * v * sum(v * h)`. It is static for the derived handle; live replies still update only existing additive steering. Supported sites are dense Llama MLP/post-output-projection attention and Qwen2 MLP on CPU/Metal and the optional Linux CUDA build. Projected handles accept text generation and logits; image requests, ordinary trace and embedding calls are refused. The combined budget includes the current artifact's validation and all accumulated projection/residual owners. Direct steer/ablate inheritance uses the projected handle's recorded budget; a checked direction application uses its explicit `max_bytes`. Extracting `direction$value` and calling `llm_steer()` directly deliberately bypasses artifact checks. Native busy/closed errors propagate.

Value

A new `llm` handle; `m` is unchanged. Close the derived handle when done.

`llm_cancel` *Cancel background generation*

Description

Requests cooperative cancellation of the async generation submitted through exactly this handle. The promise rejects with `relm_error_cancelled`; it never resolves to partial text. An in-flight native decode or image encoder call may need to finish before cancellation takes effect. The handle becomes reusable after completion is collected on R's event loop.

Usage

```
llm_cancel(m)
```

Arguments

`m`	An open `llm` handle from `llm()`.

Value

An invisible logical: TRUE only for the first accepted cancellation before native terminal publication; FALSE when idle, already cancelling, native-terminal, or when another handle submitted the active job.

See Also

llm_generate(), close.llm()

---

llm_compare *Compare actual observed states under recorded experimental settings*

Description

Builds an ordinary data frame from two delivered on_state observations. No inference is performed. Differences are withheld when the complete sampled input histories diverge. Caller-recorded context is validated, not authenticated.

Usage

```
llm_compare(
  reference,
  intervention,
  context,
  layer,
  component = "residual",
  neurons = NULL,
  max_bytes = 64 * 1024^2
)

## S3 method for class 'relm_comparison'
plot(x, ...)
```

Arguments

reference, intervention	Single observations from llm_generate(on_state=).
context	A list with reference and intervention entries, each holding model_sha256, settings, and integer generated_tokens through its state. Settings contain, in order, seed, chat, temperature, top_p, max_tokens, stop, context_length, backend, relm_version, engine_revision.
layer	One 1-based captured transformer layer.
component	One of residual, attn_out, mlp_out.
neurons	NULL for all captured coordinates, or unique 1-based indices.
max_bytes	Working materialization budget, whole bytes from 64 KiB to 256 MiB.
x	A relm_comparison returned by llm_compare().
...	Named main, cex or col (one or three colors).

Details

The input for state k excludes its just-sampled token. Equal model/settings/ prompt records establish prompt identity; every earlier sampled token must also match. A divergent history keeps the observations but returns NA differences. Missing top-k entries are NA, not zero probabilities. A sampled token may never appear in committed output after cancellation or stop removal. Returning a coefficient to zero does not erase prior effects in KV history.

Value

A `relm_comparison` data frame with layer, component, neuron, reference, intervention and difference. Attributes retain bounded provenance, alignment, applied steering and a truncated logit comparison, never the original traces.

The plot method returns the comparison invisibly.

See Also

`llm_timeline()`, `llm_generate()`

Examples

```
# Synthetic schema example: no model, inference or download.
source(system.file("examples", "graphics-states.R", package = "relm"), local = TRUE)
comparison <- llm_compare(reference, intervention, context, layer = 2)
comparison$difference
figure <- tempfile(fileext = ".pdf")
grDevices::pdf(figure, width = 10, height = 7)
tryCatch(plot(comparison), finally = grDevices::dev.off())
unlink(figure)
```

`llm_direction` *Construct a recorded contrast direction*

Description

Build a unit full-width direction from paired, already collected activations. The positive sign is target minus control. This function performs no inference.

Usage

```
llm_direction(
  target,
  control,
  context,
  layer,
  normalize_pairs = FALSE,
  orthogonalize = FALSE,
  max_bytes = 64 * 1024^2
```

```
)

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

Arguments

target, control	Plain double matrices with identical named pairs as rows and complete, ordered neuron names "1" through the hidden width as columns.
context	For construction, a recorded list with model, capture, pairs, splits and seed; see Details and the contrast-directions vignette. For application, an independently recorded destination model record.
layer	One whole layer: 2 through the recorded layer count for residual captures; 1 through that count for mlp_out and attn_out captures.
normalize_pairs	Whether to give each pair difference unit length before averaging. Defaults to FALSE, preserving relative pair magnitudes.
orthogonalize	Whether to remove the component parallel to the mean original control vector before final normalization. Defaults to FALSE.
max_bytes	Whole materialization budget, 1 MiB through 512 MiB; default 64 MiB. Includes input matrices and metadata, working arrays and output.
x	A trusted relm_direction artifact.
...	Must be empty.

Details

Context records a model checksum, architecture, quantization, hidden width, layer count and patched engine revision. Capture is full-width residual, mlp_out or attn_out space at the last prompt position, raw text, embedded tokenizer, add_special = TRUE, parse_special = FALSE, no external template, plus resolved backend, context length and package version. Pairs record ordered IDs, exact UTF-8 prompt checksums and source positions. The split manifest declares disjoint, nonempty construction, selection and evaluation sets before fitting.

At most 4096 pairs, 65536 coordinates, 16384 split rows and 8 MiB of context are admitted. Computation is row-wise; a whole difference matrix is never materialized. Zero, nearly cancelled or nonfinite directions are errors, not silently dropped observations. Norm guards use 64 * .Machine$double.eps.

Trusted saveRDS() and readRDS() preserve the artifact. Application and printing validate schema, unit values, coordinates and canonical checksums. Checksums use one bounded temporary file at a time, removed even on error. They detect modification but do not authenticate recorded provenance or already-loaded weights. Caller deserialization and native model/context memory are outside this budget. Never read untrusted RDS files. Residual artifacts retain schema 1. Component artifacts use schema 2, with component-specific identity and a versioned prefix in every checksum domain. Construction does not prove that a model supports editing the recorded site.

Value

llm_direction() returns a relm_direction data frame with integer neuron, double value and a versioned direction metadata attribute. No model pointer, input matrix or trace is retained.

See Also

`llm_trace()`, `llm_steer()`, `llm_apply_direction()`, `llm_compare()`, `llm_timeline()`

`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.
- `"qwen2-vl-2b-instruct-q4_k_m"` — Qwen2-VL 2B Instruct, Q4_K_M (Apache-2.0); the vision default. Pair it with the projector alias below: `llm(path, projector = )`.
- `"qwen2-vl-2b-instruct-mmproj-f16"` — the mmproj (vision projector) companion of the alias above (Apache-2.0, same pinned revision); pass its path as `llm()`'s `projector`.

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,
  images = NULL
)
```

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 (an empty string is allowed only for an input that carries at least one image). 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.
images	NULL (default: text-only, unchanged) or the image file paths to pair with each input — a list parallel to x, or a bare character vector for a single input. Accepted formats: JPEG, PNG, BMP. Requires a handle loaded with llm(projector =); see the *Image input* section.

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.

Image input (vision models)

On a handle loaded with llm()'s projector argument, images embeds each (text, image) pair into one row: a **list parallel to** x, where images[[i]] is a character vector of image file paths for input i (character(0) for none), or a bare character vector for a single input (recycled across several inputs with a warning; an empty vector is the same as NULL) — the same pairing contract as llm_generate()'s images. Images are inserted **before** the text. The accepted formats and pre-decode limits are exactly llm_generate()'s: **JPEG, PNG, BMP** only, at most 64 MB per file by default (options(relm.image_max_bytes =)), dimensions 1–16384 px per side, at most 33554432 total pixels; the literal marker "<__media__>" is reserved in an image-bearing input (relm_error_argument). An input that carries an image may have empty text (x = "") — the image alone is embedded; empty text without an image is still rejected.

Pooling semantics with images: the per-token vectors reduced by pooling are those of the **text positions** (including the model's own image-delimiter tokens); the image conditions those vectors through attention. This matches the reference llama.cpp behavior at the pinned engine version — image patch positions expose no per-token hidden states — and text-only inputs are completely unchanged. Images on a handle loaded without a projector raise relm_error_image.

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)


# Image embeddings (requires a projector -- see llm()).
m <- llm(Sys.getenv("RELM_TEST_MODEL_VLM"),
  projector = Sys.getenv("RELM_TEST_MMPROJ_VLM")
)
img <- file.path(tempdir(), "square.png")
png(img, width = 224, height = 224)
par(mar = c(0, 0, 0, 0))
plot.new()
rect(0.3, 0.3, 0.7, 0.7, col = "red", border = NA)
dev.off()
e <- llm_embed(m, "", images = img) # the image alone, one row
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,
  images = NULL,
  schema = NULL,
  async = FALSE,
  on_progress = NULL,
  on_token = NULL,
  on_state = NULL,
  layers = integer(),
  components = "residual",
  top = 20L,
  spill = TRUE,
  spill_dir = 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.
images	NULL (default: text-only, unchanged) or the image file paths to attach to each prompt — a list parallel to prompt, or a bare character vector for a single prompt. Accepted formats: JPEG, PNG, BMP. Requires a handle loaded with llm(projector =); see the *Image input* section.
schema	NULL (default) or one non-NA UTF-8 string containing JSON Schema text in the supported profile. This is JSON text, not a file path or R list; one schema applies to every prompt. See *Structured output*.
async	One nonmissing logical. FALSE returns synchronously; TRUE returns a promises promise backed by native background generation.
on_progress	NULL or a function accepting one progress data frame. Requires async = TRUE; see *Background generation*.
on_token	NULL, a function accepting one event data frame, or an already-open writable binary base file() connection. Requires async = TRUE; see *Token streaming*.
on_state	NULL or a function receiving one live state; requires background text-only generation. Return NULL to continue or a coefficient reply for existing steering entries; see *Live state observation*.
layers	Live capture block indices: integer() for logits only, NULL for all blocks, or unique valid 1-based indices.
components	Live capture components, using llm_trace()'s meanings.
top	Number of raw logit summaries per live state, from 0 through min(128, vocabulary size).
spill	Whether over-budget live activations may spill to Arrow files.
spill_dir	NULL for managed session storage, or a custom directory. Capture arguments require a non-NULL on_state.

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.

Without a schema, 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.

With the supported Spark-X2.5 chat template, ordinary chat opens the response with the author's <think> marker. Generated reasoning text and closing markers are returned as generated and count

toward max_tokens. With a schema, chat instead uses the author's </think> non-thinking opener, so the JSON constraint starts at the first generated token. This does not strip or repair model output. Spark chat supports the pinned official template and the existing single user turn; custom templates require chat = FALSE and caller-supplied formatting. Other models keep their existing chat behavior.

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"). With async = TRUE, a promise resolving to that vector.

Background generation

async = TRUE requires optional packages later (>= 1.4.8) and promises (>= 1.5.0), and returns a promise resolving to the same named character vector and seed attribute. Generation runs on a native worker using the loaded model. Only one native job may run per R process; competing native operations raise relm_error_busy. Ordinary R calculations and metadata printing remain usable. R validation and admission precede the omitted-seed draw. Execution errors reject the promise with their classed conditions. Use llm_cancel() to request cooperative cancellation, or close() to close the handle.

Optional on_progress(state) runs on R's thread with a fresh one-row data.frame: integer columns prompt_id, prompts_completed, prompts_total, generated_tokens, max_tokens, then character phase ("prefill", "generate", "complete"). Snapshots are coalesced and may skip intermediate states; generated tokens count the current prompt, including removed stop suffixes. Success delivers one final complete snapshot before resolving. A callback error requests cancellation and rejects with relm_error_callback, retaining the original condition in parent. Callback return values are ignored.

One timer checks progress at a 50 ms target interval on the global event loop. Callbacks normally run at the interactive prompt. In scripts, attach handlers with promises::then() and pump later::run_now() until your handler records completion. A long R expression delays delivery; a slow callback can block R. Dropping the promise or handle does not cancel a pending job. Namespace shutdown requests cancellation and joins the worker. The native library stays mapped so retained external-pointer finalizers can run safely. Forced dyn.unload() while external pointers remain alive is unsupported.

Async accepts at most 128 prompts, 1 MiB UTF-8 per prompt, 16 MiB total copied text (including names, stop strings, schema and image paths), and 8192 tokens per prompt. A separate 16 MiB descriptor budget bounds retained string/vector overhead, including empty strings. Output is limited to 8 MiB UTF-8 per call (relm_error_oom); existing stricter schema and image limits also apply. Nothing is truncated or retried to satisfy these bounds. Native cancellation is cooperative and may wait for an in-flight decode or image encoder call to finish. The 8 MiB output cap measures UTF-8 text bytes, not resident memory: copied inputs, templates, token vectors, transient output buffers and R character objects need additional storage. Existing model/context, sampler/grammar and vision working memory are separate from these transport limits.

Token streaming

Supply `on_token = function(batch) ...` to receive nonempty plain data frames on R's thread. Each dispatch contains at most 64 rows and 64 KiB of text. The columns, in order, are integer `event_id`, character `event`, integer `prompt_id`, integer `token_pos`, integer `token_id`, character `text`, double `elapsed`, character `finish_reason`, and logical `validated`. Event IDs are contiguous across the call; prompt IDs and vocabulary token IDs are 1-based.

`event = "token"` records each sampled non-EOG token, including tokens later removed with a stop suffix or sampled at context exhaustion. `token_pos` is its generated position. `event = "text"` carries a nonempty committed UTF-8 delta; concatenating these per prompt reproduces the successful returned string exactly. Tokens and text chunks need not correspond one-to-one. `event = "prompt_end"` records `finish_reason` (`"length"`, `"stop"`, `"stop_string"`, or `"context_full"`). Non-token rows have missing token fields; non-text rows have empty `text`; non-end rows have empty `finish_reason`. `elapsed` is nonnegative monotonic production time in seconds since submission. `validated` is missing for ordinary generation, `FALSE` for structured token/text, and `TRUE` for a successful structured prompt end. Structured text is provisional until that prompt's independent schema validation succeeds; this validates format, not factual accuracy.

Alternatively, supply a writable binary base `file()` connection to an empty regular local file at position zero. Append, text, compressed, raw, socket, pipe and device connections are rejected. relm neither opens nor closes the connection. Do not independently seek, write, truncate or replace its file until settlement. CSV contains one header, UTF-8 bytes, LF delimiters, comma separators and no row names. Character fields are always quoted; embedded quotes are doubled. Missing numeric/logical fields are empty; elapsed uses round-trip numeric precision independently of print options. Read it with:

```
read.csv(path, fileEncoding = "UTF-8", check.names = FALSE,
  na.strings = character(),
  colClasses = c("integer", "character", "integer", "integer", "integer",
                 "character", "numeric", "character", "logical"))
```

The connection is flushed after batches and before success; no atomic commit or disk-durability guarantee is made. Failed calls may leave a delivered prefix or an incomplete record. Successful settlement establishes completion.

Streaming retains the execution reservation until all batches are delivered. Native operations from every `on_token`, including the last, see busy; final `on_progress` runs after release. Closing the model before release abandons delivery and rejects an otherwise successful result with `relm_error_cancelled`, reason `stream_closed`. A token/progress callback error or interrupt rejects with `relm_error_callback`, identifying `callback` and preserving the original `parent`. Connection and representation failures use `relm_error_stream`, with `reason`, `prompt_id`, `event_id` and `parent` when available. The first consumer failure takes precedence over subsequent native errors; once a native failure is observed, remaining events are discarded.

A slow consumer applies backpressure to a queue bounded at 256 rows and 256 KiB text, with text chunks at most 16 KiB. Callbacks and file writes run synchronously on R and can block it. Batch boundaries and elapsed times vary across runs. Consumer side effects remain on failure, and caller-retained batches are outside relm's memory estimate. `on_token = NULL` retains the ordinary async behavior without a stream queue.

Live state observation

`on_state = function(state) ...` observes each sampled non-EOG token on the same generation context. It requires `async = TRUE`, one text prompt, `max_tokens <= 1024`, no images and no schema. The named list contains `step`, `logits` and `trace`; return `invisible(NULL)` to continue. Call `llm_cancel()` from the callback to stop before another token is sampled. Cancellation still rejects the promise, and callbacks can block R while they run.

`step` is one plain data-frame row: integer `state_id`, `prompt_id`, `token_pos`, `token_id`, `context_pos`, `source_pos`, character `source`, double `elapsed`, followed by integer `steering_revision`, `applied_after_state` and `effective_source_pos`. Let P be the actual templated prompt token count and k the generated position. The sampled token belongs at `context_pos = P + k`; the forward pass selecting it came from `source_pos = P + k - 1`. `source` is `"prompt"` for the first state and `"generated"` thereafter. Positions and vocabulary IDs are 1-based. A last sampled token may remain undecoded at a context/stop boundary; the state describes its valid source.

`logits` has `llm_logits()`'s six columns, raw logits and full-vocabulary softmax probabilities before temperature/top-p sampling. `top = 0` disables this table. `trace` is a bounded `relm_trace` for one source position, with `position_space = "model_context"`, `prompt_token_count` and `state_id` attributes. Its token positions identify the source, not the sampled token. `layers = integer()` captures no activations; `NULL` explicitly selects all blocks. Selected residuals include interventions already applied to the handle.

A callback may instead return exactly `list(steer = data.frame(intervention = 1L, coef = 0))` to change coefficients of existing steering entries. Indices address `m$interventions` (including intervening ablation entries); only steering entries may be updated. Partial replies retain other coefficients; an empty table or unchanged values do not advance the revision. Replies are validated atomically, including finite float32-range coefficients and finite summed layer vectors. Invalid replies reject with `relm_error_callback`, `callback = "on_state"`, `reason = "state_reply"`, preserving the original condition as `parent`.

A changed reply to state k affects the decode of token k and sampling of token k+1, never the already sampled token k or historical KV entries. `attr(state, "steering")` is a worker-produced data frame with integer `intervention`, integer `layer` and double `coef`, sorted by original index. Its coefficients and the three audit columns describe the adapters actually used for the reported state. Baseline audit values are 0, 0 and 1. A change from state k advances the revision and becomes effective at source position P+k; a terminal reply has no promised later-state receipt. Zero removes that entry's contribution. Ablation still follows steering.

The original adapters are restored before model ownership returns, including cancellation and failure. The R handle and other handles are unchanged; a subsequent fresh generation uses the original coefficients. Changing back during a running generation does not undo its earlier effects on KV history. Direction, layer and ablation changes are unavailable in this reply protocol.

One state waits for acknowledgement. All earlier token events are delivered before `on_state`; the current token/text events and the next decode follow the callback. No state is emitted for EOG. Removed stop-suffix tokens still have states. Inside the callback the handle remains busy; other native model operations are unavailable, while pure R calculations and lazy trace reads remain possible. A 5 ms polling target replaces the ordinary 50 ms target only for live calls; delivery depends on R's event loop.

State materialization is bounded by the smaller of `relm.trace_budget` and 32 MiB, including metadata and conversion accounting. Oversized activations spill to completed Arrow files when `spill = TRUE`; otherwise admission raises `relm_error_oom`. A single activation vector is limited to

1 MiB f32, capture/writer transport to 8 MiB plus accounted scratch, and conservative whole-call spill output to 2 GiB / 1024 files. Bounds do not cover model/KV memory, allocator overhead or objects retained/copied by user code. Managed files survive job/model closure until session cleanup; custom directories remain caller-managed. Callback failures retain their original condition in `relm_error_callback`, with `callback = "on_state"`.

Structured output

Supply `schema` as JSON text to constrain text generation to a bounded subset of JSON Schema 2020-12. The root must be a non-nullable object with explicit `properties`, `required` listing every property exactly once, and `additionalProperties: false`. Nested objects follow the same rules. Supported values are bounded strings (`maxLength`, optional `minLength`), non-nullable string enums, integers with explicit `minimum` and `maximum` in the signed 32-bit range, booleans, null, and nullable versions of these types. Nullable enums, arrays, fractional numbers and all other keywords (including annotations) are rejected. Optional root `$schema` must be `"https://json-schema.org/draft/2020-12/schema`

The schema is compiled once per call; each prompt uses fresh grammar state and the same scalar seed. Every successful element is complete, independently validated JSON text. Keys are generated in UTF-8 lexicographic order. String lengths count decoded Unicode scalar values; duplicate keys and invalid Unicode are rejected. Describe the task and fields in the prompt: a schema constrains output structure, not factual correctness.

Nonempty `stop` or image collections cannot be combined with a schema. Empty collections count as absent, and vision handles may make text-only structured requests. Generation returns when the root object is complete, including on the last allowed token. It does not retry, repair or return incomplete JSON. Malformed, unsupported or oversized schemas raise `relm_error_schema` with `reason` and `schema_path` (a JSON pointer). Generation failure raises `relm_error_structured_output` with `reason`, 1-based `prompt_id`, `seed`, `generated_tokens` and bounded `partial_bytes` (a raw vector). The first failed prompt aborts the call; no partial result vector is returned. Input context overflow retains `relm_error_context_overflow`.

Hard limits: schema text 64 KiB, nesting 8, schema nodes 128, 16 properties per object and 64 total; 32 enum members of at most 128 Unicode scalar values; string `maxLength` 2048; compiled grammar 512 KiB and 65536 elements. A call accepts at most 128 prompts, 1 MiB per prompt and 16 MiB total UTF-8 input, with `max_tokens <= 8192`. Temperature must be finite and fit a 32-bit float; an explicit seed must be finite and less than `2^64` (subject to R's numeric precision). Output is limited to 64 KiB per prompt and 8 MiB per call. `schema = NULL` preserves ordinary text and vision generation.

Image input (vision models)

On a handle loaded with `llm()`'s `projector` argument, `images` attaches image files to each prompt: a **list parallel to** `prompt`, where `images[[i]]` is a character vector of image file paths for prompt `i` (`character(0)` for none). A bare character vector is treated as `list(images)` — one image set — and pairs with a single prompt; with several prompts it is recycled across all of them with a warning (an empty vector is the same as `NULL`: no images, no warning). Each prompt's images are inserted **before** its text. Exactly three file formats are accepted: **JPEG, PNG, BMP** (anything else — GIF and audio included — is rejected before any decode with `relm_error_image`). Size limits, enforced before decoding: at most 64 MB per file by default (override with `options(relm.image_max_bytes = )`; hard ceiling 2147483647 bytes), each dimension between 1 and 16384 pixels, and at most 33554432 total pixels. Images on a handle loaded without a projector raise `relm_error_image`;

the combined text+image token count must fit `context_length` (`relm_error_context_overflow` states by how much).

One content restriction applies to an image-bearing prompt: the literal string `"<__media__>"` (the engine's internal media marker) is reserved — relm inserts one marker per image before the text, so a literal marker in the prompt would corrupt the image placement, and the call raises `relm_error_argument` naming `prompt`. Prompts without images may contain the string freely (it is ordinary text there).

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)
schema <- paste0('{"type":"object","properties":{"answer":',
  '{"type":"string","enum":["yes","no"]}},',
  '"required":["answer"],"additionalProperties":false}')
llm_generate(m, 'Is R a programming language? Return an object with "answer".',
  schema = schema, max_tokens = 64, temperature = 0, seed = 1)
close(m)


# Image input (requires a projector -- see llm()).
m <- llm(Sys.getenv("RELM_TEST_MODEL_VLM"),
  projector = Sys.getenv("RELM_TEST_MMPROJ_VLM")
)
img <- file.path(tempdir(), "square.png")
png(img, width = 224, height = 224)
par(mar = c(0, 0, 0, 0))
plot.new()
rect(0.3, 0.3, 0.7, 0.7, col = "red", border = NA)
dev.off()
llm_generate(m, "What color is the square?",
  images = img,
  max_tokens = 16, temperature = 0
)
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_probe` *Fit grouped statistical probes to activation traces*

Description

Fits one binary ridge logistic probe per layer using the optional glmnet package. Group-disjoint development cross-validation selects regularization and the default layer. Scaling and fitted coefficients use development data only. A declared holdout evaluates frozen fits; absent a holdout, results are exploratory CV diagnostics without inferential confidence intervals.

Usage

```
llm_probe(
  formula,
  data,
  method = "glmnet",
  cv = 10,
  metric = c("auc", "accuracy"),
  seed = NULL,
  groups = NULL,
  test_groups = NULL
)
```

Arguments

`formula`	A formula such as `label ~ activations(layer = 1:3)`. Labels are logical, numeric 0/1, or a two-level factor (second level positive), from a prompt-constant trace column or an external vector in the formula environment.
`data`	A `relm_trace`, with one captured position per prompt, matching prompt/position/neuron coordinates across requested layers. Spilled traces require an external label vector.
`method`	Currently only `"glmnet"` (binary ridge logistic regression).
`cv`	Number of group-disjoint development folds, an integer at least 3. Infeasible class support raises an error rather than reducing this value.
`metric`	Selection and reporting metric: `"auc"` or `"accuracy"`. Accuracy uses a fixed probability threshold of 0.5.
`seed`	Optional nonnegative integer. Seeded calls restore the caller's RNG state, including its prior absence; `NULL` uses the caller's RNG.
`groups`	Character source-group IDs, one per captured prompt in increasing prompt-ID order, or named by exact prompt IDs. `NULL` assumes each prompt is independent. Related documents, revisions, paraphrases and matched contrasts must share a caller-supplied group.
`test_groups`	Character IDs of whole groups reserved before analysis, or `NULL` for exploratory development CV without held-out estimates.

Details

The fixed ridge grid is `10^seq(4, -4, length.out = 41)`. Each fitting partition supplies its own means and RMS scales. Constant features are dropped. Selection uses the mean of fold metrics, with stronger regularization and then the lower layer breaking ties. Errors inherit `relm_error_probe`; an estimated materialized-memory excess raises `relm_error_oom` before densification, using `options(relm.trace_budget` as a ceiling.

Held-out intervals use 2,000 whole-group bootstrap draws and type-7 percentile quantiles. They are approximate 95% pointwise intervals for sampling variability conditional on the frozen fits, development data, selection and split, not training/selection uncertainty or simultaneous coverage. Intervals are withheld with a reason below 20 evaluation groups, below five groups containing either class, on undefined draws, or for a degenerate interval. Larger groups contribute more observations to metrics. Decodability is association, not evidence of causal use. Repeatedly changing an analysis after inspecting the holdout consumes its independent status.

Value

An `llm_probe` classed list: development-only `fits`, per-layer `metrics`, fold-level `cv_scores`, prompt-level split `audit`, held-out `predictions`, `selected_layer`, counts and statistical provenance. Fits store compact coefficients and preprocessing, not every CV path or the source trace.

See Also

`llm_trace()`, `activations()`, `predict.llm_probe()`

Examples

```
# Model-free trace for demonstrating the API, not scientific evidence.
label <- rep(0:1, 12)
tr <- expand.grid(neuron = 1:2, layer = 1:2, prompt_id = seq_along(label))
tr$token_pos <- 1L
tr$token <- "example"
tr$component <- "residual"
tr$value <- sin(tr$prompt_id * tr$neuron) + label[tr$prompt_id]
class(tr) <- c("relm_trace", "data.frame")
fit <- llm_probe(label ~ activations(layer = 1:2), tr, cv = 3, seed = 42)
summary(fit)
predict(fit, tr)
```

`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). Prompt tracing runs on a dedicated transient context. Live generation uses a separate dormant dispatcher on the generation context; selected live taps activate only when `llm_generate(on_state = ...)` is requested.

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)
```

`plot.llm` *Draw a model block map and configured intervention sites*

Description

The schematic describes architecture metadata, not measured activity, causal influence or coefficients currently applied by an active worker. Unknown architectures use generic blocks; unsupported captures are labelled.

Usage

```
## S3 method for class 'llm'
plot(x, layers = NULL, ...)
```

Arguments

`x`	An open llm handle.
`layers`	NULL for up to 32 blocks, or explicit unique 1-based indices.
`...`	Named main, cex or col (one or three colors).

Value

Invisibly, the plain data frame of displayed sites and intervention counts. `configured_projections` counts static component projections, including zero coefficients; it does not count additive live revisions.

See Also

`llm_timeline()`, `llm_compare()`

Examples

```
# Optional local model; never downloads a model during examples.
path <- Sys.getenv("RELM_EXAMPLE_MODEL", "")
if (nzchar(path) && file.exists(path)) {
  model <- llm(path, backend = "cpu")
  figure <- tempfile(fileext = ".pdf")
  grDevices::pdf(figure, width = 10, height = 7)
  tryCatch(plot(model, layers = 1:2),
           finally = { grDevices::dev.off(); close(model) })
  unlink(figure)
}
```

`plot.relm_timeline` *Retain a bounded history of applied live steering*

Description

Appends the next on_state audit, dropping oldest complete states to respect the explicit working budget and retention window. No activations or model handles are retained. The table records sampled states, not committed text.

Usage

```
## S3 method for class 'relm_timeline'
plot(x, ...)

llm_timeline(state, history = NULL, max_states = 256L, max_bytes = 8 * 1024^2)
```

Arguments

`x`	A relm_timeline returned by llm_timeline(), optionally row-filtered to at most 16 intervention IDs for rendering.
`...`	Named main, cex or col (one or three colors).
`state`	A single observation delivered to `on_state`.
`history`	NULL to start at state 1, or the previous returned history.
`max_states`	Number of newest whole states to retain, from 1 through 1024.
`max_bytes`	Working materialization budget, whole bytes from 64 KiB to 64 MiB.

Details

Inside a callback assign `history <- llm_timeline(state, history)`, then explicitly return NULL or the intended steering reply. Do not return this history as a callback reply. A change requested at state k applies to the next decode, and setting a coefficient to zero does not clear KV history. The helper validates consistency of editable R records, not their origin.

Value

The plot method returns the history invisibly.

A `relm_timeline` data frame containing step columns followed by intervention, layer and coef, plus explicit retained/dropped ranges.

See Also

`llm_compare()`, `llm_generate()`

Examples

```
source(system.file("examples", "graphics-states.R", package = "relm"), local = TRUE)
history <- llm_timeline(reference, max_states = 8)
next_state <- example_state(2L, coefficient = 0, revision = 1L, after = 1L)
history <- llm_timeline(next_state, history, max_states = 8)
history[, c("state_id", "coef", "applied_after_state")]
figure <- tempfile(fileext = ".pdf")
grDevices::pdf(figure, width = 10, height = 7)
tryCatch(plot(history), finally = grDevices::dev.off())
unlink(figure)
```

---

predict.llm_probe *Predict binary probabilities from a fitted probe*

Description

Predict binary probabilities from a fitted probe

Usage

```
## S3 method for class 'llm_probe'
predict(object, newdata, layer = NULL, ...)
```

Arguments

object	An llm_probe result.
newdata	A relm_trace with the same component and neuron coordinates and one position per prompt. It can contain new prompt IDs and needs no labels.
layer	Optional fitted layer index. NULL uses the layer selected on development CV, regardless of held-out performance.
...	Ignored.

Value

A numeric probability vector for the positive class, named by prompt ID in increasing order. Uses the saved development-only transformation and fit.

Examples

```
# See llm_probe() for a complete model-free fit and prediction example.
```

probe-methods *Summarize and plot a statistical probe set*

Description

Summarize and plot a statistical probe set

Usage

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

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

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

## S3 method for class 'llm_probe'
plot(x, ...)
```

Arguments

x, object	An llm_probe result.
...	Additional base plotting arguments for plot(). Ignored by print and summary methods.

Details

Held-out curves mark the layer chosen by development CV. Their intervals are conditional and pointwise, not simultaneous or causal claims. Unsupported intervals have a recorded reason. Exploratory CV curves never receive inferential error bars. The chosen layer is not changed by plotting.

Value

print() returns its input invisibly. summary() returns a classed list of metrics, counts, split audit, preprocessing and statistical provenance. plot() invisibly returns the displayed metric table.

Examples

```
# See llm_probe() for a complete model-free example.
```

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

activations, 3
activations(), *26*
as.matrix.relm_trace, 3
as.matrix.relm_trace(), *31*

close(), *5*, *19*
close.llm, 4
close.llm(), *5*, *6*, *10*

llm, 5
llm(), *4*, *7–9*, *14–17*, *22–24*, *27*, *28*, *30*, *31*
llm_ablate, 7
llm_ablate(), *15*, *24*, *27*, *28*, *31*
llm_apply_direction, 8
llm_apply_direction(), *13*
llm_cancel, 9
llm_cancel(), *19*, *21*
llm_compare, 10
llm_compare(), *13*, *32*, *33*
llm_direction, 11
llm_direction(), *8*
llm_download, 13
llm_download(), *6*
llm_embed, 14
llm_embed(), *6*, *7*, *27*, *31*
llm_generate, 17
llm_generate(), *6*, *8*, *10*, *11*, *16*, *23*, *24*, *28*, *33*
llm_logits, 23
llm_logits(), *21*
llm_probe, 25
llm_probe(), *3*
llm_steer, 26
llm_steer(), *8*, *9*, *13*, *15*, *24*, *31*
llm_timeline *(*plot.relm_timeline*)*, 33
llm_timeline(), *11*, *13*, *32*
llm_tokens, 28
llm_tokens(), *16*, *23*, *24*
llm_trace, 29
llm_trace(), *3*, *4*, *7*, *13*, *14*, *18*, *26*, *27*

plot.llm, 32
plot.llm_probe *(*probe-methods*)*, 35
plot.relm_comparison *(*llm_compare*)*, 10
plot.relm_timeline, 33
predict.llm_probe, 34
predict.llm_probe(), *26*
print.llm *(*llm*)*, 5
print.llm(), *6*
print.llm_probe *(*probe-methods*)*, 35
print.relm_direction *(*llm_direction*)*, 11
print.relm_trace *(*llm_trace*)*, 29
print.summary.llm *(*summary.llm*)*, 36
print.summary.llm_probe *(*probe-methods*)*, 35
print.summary.relm_trace *(*llm_trace*)*, 29
probe-methods, 35

stats::prcomp(), *3*
summary.llm, 36
summary.llm(), *6*
summary.llm_probe *(*probe-methods*)*, 35
summary.relm_trace *(*llm_trace*)*, 29