# Package 'localLLM'

May 8, 2026

**Type** Package

**Title** Running Local LLMs with 'llama.cpp' Backend

**Version** 1.3.0

**Date** 2026-05-04

**Author** Eddie Yang [aut] (ORCID: <https://orcid.org/0000-0002-3696-3226>),
Yaosheng Xu [aut, cre] (ORCID: <https://orcid.org/0009-0006-8138-369X>)

**Maintainer** Yaosheng Xu <xu2009@purdue.edu>

**Description** Provides R bindings to the 'llama.cpp' library for running large language models. The package uses a lightweight architecture where the C++ backend library is downloaded at runtime rather than bundled with the package.
Package features include text generation, reproducible generation, and parallel inference.

**License** MIT + file LICENSE

**Depends** R (>= 3.6.0)

**LinkingTo** Rcpp

**Imports** Rcpp (>= 1.0.14), tools, utils, jsonlite, digest, curl, R.utils

**Suggests** testthat (>= 3.0.0), covr, irr, knitr, rmarkdown

**VignetteBuilder** knitr

**URL** https://github.com/EddieYang211/localLLM

**BugReports** https://github.com/EddieYang211/localLLM/issues

**SystemRequirements** C++17, libcurl (optional, for model downloading)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2026-05-05 05:30:02 UTC

# Contents



---

`localLLM-package` *R Interface to llama.cpp with Runtime Library Loading*

---

## Description

Provides R bindings to the llama.cpp library for running large language models locally. This package uses an innovative lightweight architecture where the C++ backend library is downloaded at runtime rather than bundled with the package, enabling zero-configuration AI inference in R with enterprise-grade performance.

## Details

The localLLM package brings state-of-the-art language models to R users through a carefully designed four-layer architecture that combines ease of use with high performance.

## Quick Start 1. Install the R package: `install.packages("localLLM")` 2. Download backend library: `install_localLLM()` 3. Start generating text: `quick_llama("Hello, how are you?")`

## Key Features

- **Zero Configuration**: One-line setup with automatic model downloading
- **High Performance**: Native C++ inference engine with GPU support
- **Cross Platform**: Pre-compiled binaries for Windows, macOS, and Linux
- **Memory Efficient**: Smart caching and memory management
- **Production Ready**: Robust error handling and comprehensive documentation

## Architecture Overview The package uses a layered design:

- **High-Level API**: `quick_llama` for simple text generation
- **Mid-Level API**: `model_load`, `generate` for detailed control
- **Low-Level API**: Direct access to tokenization and context management
- **C++ Backend**: llama.cpp engine with dynamic loading

## Main Functions

- `install_localLLM` - Download and install backend library
- `quick_llama` - High-level text generation (recommended for beginners)
- `model_load` - Load GGUF models with smart caching
- `context_create` - Create inference contexts
- `generate` - Generate text with full parameter control
- `tokenize` / `detokenize` - Text <-> Token conversion
- `apply_chat_template` - Format conversations for chat models

## Example Workflows

### Basic Text Generation

```
# Simple one-liner
response <- quick_llama("Explain quantum computing")

# With custom parameters
creative_text <- quick_llama("Write a poem about AI",
                             temperature = 0.9,
                             max_tokens = 150)
```

### Advanced Usage with Custom Models

```
# Load your own model
model <- model_load("path/to/your/model.gguf")
ctx <- context_create(model, n_ctx = 4096)

# Direct text generation with auto-tokenization
output <- generate(ctx, "The future of AI is", max_tokens = 100)
```

### Batch Processing

```
# Process multiple prompts efficiently
prompts <- c("Summarize AI trends",
             "Explain machine learning",
             "What is deep learning?")
responses <- quick_llama(prompts)
```

## Supported Model Formats The package works with GGUF format models from various sources:

- Hugging Face Hub (automatic download)
- Local .gguf files
- Custom quantized models
- Ollama-compatible models

## Performance Tips

- Use `n_gpu_layers = -1` to fully utilize GPU acceleration
- Set `n_threads` to match your CPU cores for optimal performance
- Use larger `n_ctx` values for longer conversations
- Enable `use_mlock` for frequently used models to prevent swapping

## Author(s)

Eddie Yang and Yaosheng Xu <xu2009@purdue.edu>

## References

https://github.com/EddieYang211/localLLM

## See Also

Useful links:

- https://github.com/EddieYang211/localLLM
- Report bugs at https://github.com/EddieYang211/localLLM/issues

## ag_news_sample *AG News classification sample*

### Description

A 100-row subset of the AG News Topic Classification dataset consisting of 25 documents from each of the four classes (World, Sports, Business, Sci/Tech). The sample is intended for quick demonstrations and tests without requiring the full external dataset.

### Usage

```
data(ag_news_sample)
```

### Format

A data frame with 100 rows and 3 character columns:

**class** News topic label (`"World"`, `"Sports"`, `"Business"`, or `"Sci/Tech"`).

**title** Headline of the news article.

**description** Short description for the article.

### Details

The sample was obtained from `textdata::dataset_ag_news()` (Zhang et al., 2015) using a fixed random seed to ensure reproducibility. It is provided solely for illustrative purposes.

### Source

Zhang, X., Zhao, J., & LeCun, Y. (2015). "Character-level Convolutional Networks for Text Classification." arXiv:1509.01626. Original data distributed via the AG News Topic Classification dataset.

### See Also

[textdata::dataset_ag_news()]

## annotation_sink_csv *Create a CSV sink for streaming annotation chunks*

### Description

The returned closure can be passed to ‘explore(sink = ...)‘ to append each per-model chunk to a CSV file without holding everything in memory.

### Usage

```
annotation_sink_csv(path, append = FALSE)
```

### Arguments

| | |
|---|---|
| path | Destination CSV path. |
| append | If 'TRUE', new chunks are appended to an existing file. |

### Value

A function with signature '(chunk, model_id)'.

---

## apply_chat_template *Apply Chat Template to Format Conversations*

---

### Description

Formats conversation messages using the model's built-in chat template or a custom template. This is essential for chat models that expect specific formatting for multi-turn conversations.

### Usage

```
apply_chat_template(model, messages, template = NULL,
  add_assistant = TRUE)
```

### Arguments

| | |
|---|---|
| model | A model object created with model_load |
| messages | List of chat messages, each with 'role' and 'content' fields. Role should be 'user', 'assistant', or 'system' |
| template | Optional custom template string (default: NULL, uses model's built-in template) |
| add_assistant | Whether to add assistant prompt suffix for response generation (default: TRUE) |

### Value

Formatted prompt string ready for text generation

### See Also

model_load, quick_llama, generate

### Examples

```
## Not run:
# Load a chat model
model <- model_load("path/to/chat_model.gguf")

# Format a conversation
messages <- list(
  list(role = "system", content = "You are a helpful assistant."),
```

```
  list(role = "user", content = "What is machine learning?"),
  list(role = "assistant", content = "Machine learning is..."),
  list(role = "user", content = "Give me an example.")
)

# Apply chat template
formatted_prompt <- apply_chat_template(model, messages)

# Generate response
response <- quick_llama(formatted_prompt)

## End(Not run)
```

---

`apply_gemma_chat_template` *Apply Gemma-Compatible Chat Template*

---

## Description

Creates a properly formatted chat template for Gemma models, which use <start_of_turn> and <end_of_turn> markers instead of ChatML format. This function addresses compatibility issues with apply_chat_template() when used with Gemma models.

## Usage

```
apply_gemma_chat_template(messages, add_assistant = TRUE)
```

## Arguments

| | |
|---|---|
| `messages` | A list of message objects, each with 'role' and 'content' fields |
| `add_assistant` | Whether to add the assistant turn prefix (default: TRUE) |

## Value

A character string with properly formatted Gemma chat template

## Examples

```
## Not run:
messages <- list(
  list(role = "system", content = "You are a helpful assistant."),
  list(role = "user", content = "Hello!")
)
formatted <- apply_gemma_chat_template(messages)

## End(Not run)
```

## backend_free *Free localLLM backend*

### Description

Clean up backend resources. Usually called automatically.

### Usage

```
backend_free()
```

### Value

No return value, called for side effects (frees backend resources).

## backend_init *Initialize localLLM backend*

### Description

Initialize the backend library. This should be called once before using other functions.

### Usage

```
backend_init()
```

### Value

No return value, called for side effects (initializes backend).

## compute_confusion_matrices *Compute confusion matrices from multi-model annotations*

### Description

Compute confusion matrices from multi-model annotations

### Usage

```
compute_confusion_matrices(
  annotations,
  gold = NULL,
  pairwise = TRUE,
  label_levels = NULL,
  sample_col = "sample_id",
  model_col = "model_id",
  label_col = "label",
  truth_col = "truth"
)
```

### Arguments

| | |
|---|---|
| annotations | Output from [explore()] or a compatible data frame with at least 'sample_id', 'model_id', and 'label' columns. |
| gold | Optional vector of gold labels. Overrides the 'truth' column when supplied. |
| pairwise | When 'TRUE', cross-model confusion tables are returned even if no gold labels exist. |
| label_levels | Optional factor levels to enforce a consistent ordering in the resulting tables. |
| sample_col, model_col, label_col, truth_col | Column names to use when 'annotations' is a custom data frame. |

### Value

A list with elements 'vs_gold' (named list of matrices, one per model) and 'pairwise' (list of pairwise confusion tables).

---

## context_create *Create Inference Context for Text Generation*

---

### Description

Creates a context object that manages the computational state for text generation. The context maintains the conversation history and manages memory efficiently for processing input tokens and generating responses. Each model can have multiple contexts with different settings.

### Usage

```
context_create(
  model,
  n_ctx = 2048L,
  n_threads = 4L,
  n_seq_max = 1L,
  verbosity = 1L
)
```

## Arguments

| | |
|---|---|
| model | A model object returned by model_load |
| n_ctx | Maximum context length in tokens (default: 2048). This determines how many tokens of conversation history can be maintained. Larger values require more memory but allow for longer conversations. Must not exceed the model's maximum context length |
| n_threads | Number of CPU threads for inference (default: 4). Set to the number of available CPU cores for optimal performance. Only affects CPU computation |
| n_seq_max | Maximum number of parallel sequences (default: 1). Used for batch processing multiple conversations simultaneously. Higher values require more memory |
| verbosity | Control backend logging during context creation (default: 1L). Larger values print more information: 0 emits only errors, 1 includes warnings, 2 adds informational logs, and 3 enables the most verbose debug output. |

## Value

A context object (external pointer) used for text generation with generate

## See Also

model_load, generate, tokenize

## Examples

```
## Not run:
# Load model and create basic context
model <- model_load("path/to/model.gguf")
ctx <- context_create(model)

# Create context with larger buffer for long conversations
long_ctx <- context_create(model, n_ctx = 4096)

# High-performance context with more threads
fast_ctx <- context_create(model, n_ctx = 2048, n_threads = 8)

# Context for batch processing multiple conversations
batch_ctx <- context_create(model, n_ctx = 2048, n_seq_max = 4)

# Create context with minimal verbosity (quiet mode)
quiet_ctx <- context_create(model, verbosity = 2L)

## End(Not run)
```

detokenize *Convert Token IDs Back to Text*

## Description

Converts a sequence of integer token IDs back into human-readable text. This is the inverse operation of tokenization and is typically used to convert model output tokens into text that can be displayed to users.

## Usage

```
detokenize(model, tokens)
```

## Arguments

| | |
|---|---|
| model | A model object created with model_load. Must be the same model that was used for tokenization to ensure proper decoding |
| tokens | Integer vector of token IDs to convert back to text. These are typically generated by tokenize or generate |

## Value

Character string containing the decoded text corresponding to the input tokens

## See Also

tokenize, generate, model_load

## Examples

```
## Not run:
# Load model
model <- model_load("path/to/model.gguf")

# Tokenize then detokenize (round-trip)
original_text <- "Hello, how are you today?"
tokens <- tokenize(model, original_text)
recovered_text <- detokenize(model, tokens)
print(recovered_text)  # Should match original_text

# Generate and display text
ctx <- context_create(model)
generated_text <- generate(ctx, "The weather is", max_tokens = 10)

# Inspect individual tokens
single_token <- c(123)  # Some token ID
token_text <- detokenize(model, single_token)
print(paste("Token", single_token, "represents:", token_text))

## End(Not run)
```

## document_end *Finish automatic run documentation*

### Description

Flushes the buffered log entries assembled since the matching [document_start()] call and writes them to the configured text file. A SHA-256 hash of the written content is appended to the log so runs can be compared or referenced succinctly.

### Usage

```
document_end()
```

### Value

Invisibly returns the log file path with attribute ‘hash‘ containing the SHA-256 digest of the run contents.

## document_start *Start automatic run documentation*

### Description

Calling ‘document_start()‘ enables automatic logging for subsequent ‘localLLM‘ calls. Information such as timestamps, models, and generation settings are buffered in-memory until [document_end()] is invoked, at which point a human-readable text report is written to disk.

### Usage

```
document_start(path = NULL, metadata = list(), append = FALSE)
```

### Arguments

| | |
|---|---|
| path | Optional log file path. Defaults to ‘localLLM_run_<timestamp>.txt‘ in the working directory. |
| metadata | Optional named list of user-defined metadata to include in the log header (e.g. project name, dataset id). |
| append | When ‘TRUE‘, entries are appended to an existing file instead of overwriting it. |

### Value

The path that will be written when [document_end()] is called.

download_model *Download a model manually*

## Description

Download a model manually

## Usage

```
download_model(
  model_url,
  output_path = NULL,
  show_progress = TRUE,
  verify_integrity = TRUE,
  max_retries = 3,
  hf_token = NULL
)
```

## Arguments

| | |
|---|---|
| model_url | URL of the model to download (currently only supports https://) |
| output_path | Local path where to save the model (optional, will use cache if not provided) |
| show_progress | Whether to show download progress (default: TRUE) |
| verify_integrity | Verify file integrity after download (default: TRUE) |
| max_retries | Maximum number of download retries (default: 3) |
| hf_token | Optional Hugging Face access token to use for this download. Defaults to the existing 'HF_TOKEN' environment variable. |

## Value

The path where the model was saved

## Examples

```
## Not run:
# Download to specific location
download_model(
  "https://example.com/model.gguf",
  file.path(tempdir(), "my_model.gguf")
)

# Download to cache (path will be returned)
cached_path <- download_model("https://example.com/model.gguf")

## End(Not run)
```

`explore` *Compare multiple LLMs over a shared set of prompts*

## Description

‘explore()‘ orchestrates running several models over the same prompts, captures their predictions, and returns both long and wide annotation tables that can be fed into confusion-matrix and reliability helpers.

## Usage

```
explore(
  models,
  instruction = NULL,
  prompts = NULL,
  engine = c("auto", "parallel", "single"),
  batch_size = 8L,
  reuse_models = FALSE,
  sink = NULL,
  progress = interactive(),
  clean = TRUE,
  keep_prompts = FALSE,
  hash = TRUE,
  chat_template = TRUE,
  system_prompt = NULL
)
```

## Arguments

`models` Model definitions. Accepts one of the following formats:

- A single model path string (consistent with [model_load()] syntax)
- A named character vector where names become ‘model_id‘s
- A list of model specification lists

Each model specification list supports the following keys:

**id** (Required unless auto-generated) Unique identifier for this model

**model_path** (Required unless using ‘predictor‘) Path to local GGUF file, URL, or cached model name. Supports the same formats as [model_load()]

**n_gpu_layers** Number of layers to offload to GPU. Use ‘"auto"‘ (default) for automatic detection, ‘0‘ for CPU-only, or ‘-1‘ for all layers on GPU

**n_ctx** Context window size (default: 2048)

**n_threads** Number of CPU threads (default: auto-detected)

**cache_dir** Custom cache directory for model downloads

**use_mmap** Enable memory mapping (default: TRUE)

**use_mlock** Lock model in memory (default: FALSE)

**check_memory** Check memory availability before loading (default: TRUE)

**force_redownload** Force re-download even if cached (default: FALSE)

**verify_integrity** Verify file integrity (default: TRUE)

**hf_token** Hugging Face access token for gated models. Can also be set globally via [set_hf_token()]

**verbosity** Backend logging level (default: 1)

**chat_template** Override the global 'chat_template' setting for this model

**system_prompt** Override the global 'system_prompt' for this model

**instruction** Task instruction to use for this model

**generation** List of generation parameters (max_tokens, temperature, etc.)

**prompts** Custom prompts for this model

**predictor** Function for mock/testing scenarios (bypasses model loading)

`instruction` Default task instruction inserted into 'spec' whenever a model entry does not override it.

`prompts` One of: (1) a function (for example 'function(spec)') that returns prompts (character vector or a data frame with a 'prompt' column); (2) a character vector of ready-made prompts; or (3) a template list where each named element becomes a section in the rendered prompt. Field names are used as-is for headers. Vector fields matching 'sample_id' length are treated as per-item values. Use 'sample_id' to specify item IDs (meta, not rendered). When 'NULL', each model must provide its own 'prompts' entry.

`engine` One of '"auto"', '"parallel"', or '"single"'. Selects whether the parallel or single-prompt backend is used.

`batch_size` Number of prompts to send per backend call when the parallel engine is active. Must be >= 1.

`reuse_models` If 'TRUE', model/context handles stay alive for the duration of the function (useful when exploring lots of prompts). When 'FALSE' (default) handles are released after each model to minimise peak memory usage.

`sink` Optional function that accepts '(chunk, model_id)' and is invoked after each model finishes. This makes it easy to stream intermediate results to disk via helpers such as [annotation_sink_csv()].

`progress` Whether to print progress messages for each model/batch.

`clean` Forwarded to 'generate()'/'generate_parallel()' to remove control tokens from the outputs.

`keep_prompts` If 'TRUE', the generated prompts are preserved in the long-format output (useful for audits). Defaults to 'FALSE'.

`hash` When 'TRUE' (default), computes SHA-256 hashes for each model's prompts and resulting labels so replication collaborators can verify inputs and outputs. Hashes are attached to the returned list via the '"hashes"' attribute.

`chat_template` When 'TRUE', wraps prompts using the model's built-in chat template before generation. This uses [apply_chat_template()] to format prompts with appropriate special tokens for instruction-tuned models. Individual models can override this via their spec. Default: 'TRUE'.

`system_prompt` Optional system message to include when 'chat_template = TRUE'. This is prepended as a system role message before the user prompt. Individual models can override this via their spec. Default: 'NULL'.

## Value

A list with elements ‘annotations‘ (long table) and ‘matrix‘ (wide annotation matrix). When ‘sink‘ is supplied the ‘annotations‘ and ‘matrix‘ entries are set to ‘NULL‘ to avoid duplicating the streamed output.

---

`generate` *Generate Text Using Language Model Context*

---

## Description

Generates text using a loaded language model context with automatic tokenization. Simply provide a text prompt and the model will handle tokenization internally. This function now has a unified API with `generate_parallel`.

## Usage

```
generate(
  context,
  prompt,
  max_tokens = 100L,
  top_k = 40L,
  top_p = 1,
  temperature = 0,
  repeat_last_n = 0L,
  penalty_repeat = 1,
  seed = 1234L,
  clean = FALSE,
  hash = TRUE,
  verbosity = 0L
)
```

## Arguments

| | |
|---|---|
| `context` | A context object created with `context_create` |
| `prompt` | Character string containing the input text prompt |
| `max_tokens` | Maximum number of tokens to generate (default: 100). Higher values produce longer responses |
| `top_k` | Top-k sampling parameter (default: 40). Limits vocabulary to k most likely tokens. Use 0 to disable |
| `top_p` | Top-p (nucleus) sampling (default: 1.0). Probability threshold for token selection. |
| `temperature` | Sampling temperature (default: 0.0). Set to 0 for greedy decoding. Higher values increase creativity |
| `repeat_last_n` | Number of recent tokens to consider for repetition penalty (default: 0). Set to 0 to disable |

penalty_repeat — Repetition penalty strength (default: 1.0). Values >1 discourage repetition. Set to 1.0 to disable

seed — Random seed for reproducible generation (default: 1234). Use positive integers for deterministic output

clean — If TRUE, strip common chat-template control tokens from the generated text (default: FALSE).

hash — When ‘TRUE‘ (default), computes SHA-256 hashes for the provided prompt and the resulting output. Hashes are attached via the ‘"hashes"‘ attribute for later inspection.

verbosity — Control backend logging during generation (default: 0L). Larger numbers print more detail: `0` shows only errors, `1` adds warnings, `2` prints informational messages, and `3` enables the most verbose debug output. Negative values fully suppress backend output. Defaults to quiet (`0`) because `generate()` is typically called inside loops where per-call backend logs would be noisy. This differs from `model_load` and `context_create` (default `1L`), which run once per session and benefit from warnings being visible. Raise to `2L` or `3L` when debugging llama.cpp internals.

## Value

Character string containing the generated text

## See Also

`quick_llama`, `generate_parallel`, `context_create`

## Examples

```
## Not run:
# Load model and create context
model <- model_load("path/to/model.gguf")
ctx <- context_create(model, n_ctx = 2048)

response <- generate(ctx, "Hello, how are you?", max_tokens = 50)

# Creative writing with higher temperature
story <- generate(ctx, "Once upon a time",
                  max_tokens = 200, temperature = 0.8)

# Prevent repetition
no_repeat <- generate(ctx, "Tell me about AI",
                      repeat_last_n = 64,
                      penalty_repeat = 1.1)

# Clean output (remove special tokens)
clean_output <- generate(ctx, "Explain quantum physics", clean = TRUE)

## End(Not run)
```

generate_parallel *Generate Text in Parallel for Multiple Prompts*

## Description

Generate Text in Parallel for Multiple Prompts

## Usage

```
generate_parallel(
  context,
  prompts,
  max_tokens = 100L,
  top_k = 40L,
  top_p = 1,
  temperature = 0,
  repeat_last_n = 0L,
  penalty_repeat = 1,
  seed = 1234L,
  progress = interactive(),
  verbosity = 0L,
  clean = FALSE,
  hash = TRUE
)
```

## Arguments

| | |
|---|---|
| context | A context object created with context_create |
| prompts | Character vector of input text prompts |
| max_tokens | Maximum number of tokens to generate (default: 100) |
| top_k | Top-k sampling parameter (default: 40). Limits vocabulary to k most likely tokens |
| top_p | Top-p (nucleus) sampling (default: 1.0). Probability threshold for token selection. |
| temperature | Sampling temperature (default: 0.0). Set to 0 for greedy decoding. Higher values increase creativity |
| repeat_last_n | Number of recent tokens to consider for repetition penalty (default: 0). Set to 0 to disable |
| penalty_repeat | Repetition penalty strength (default: 1.0). Values >1 discourage repetition. Set to 1.0 to disable |
| seed | Random seed for reproducible generation (default: 1234). Use positive integers for deterministic output |
| progress | Show a console progress bar while batches run. Defaults to interactive(): visible in interactive sessions, suppressed in scripts and R CMD check. |

verbosity — Control backend logging during generation (default: 0L). Larger numbers print more detail: `0` shows only errors, `1` adds warnings, `2` prints informational messages, and `3` enables the most verbose debug output. Negative values fully suppress backend output. Defaults to quiet (`0`) so that only the progress bar is visible during typical batch runs, matching `generate`. This differs from `model_load` and `context_create` (default `1L`), which run once per session and benefit from warnings being visible. Raise to `2L` or `3L` when debugging llama.cpp internals.

clean — If TRUE, remove common chat-template control tokens from each generated text (default: FALSE).

hash — When ‘TRUE‘ (default), computes SHA-256 hashes for the supplied prompts and generated outputs. Hashes are attached via the ‘"hashes"‘ attribute for later inspection.

## Details

When more prompts are supplied than the context can hold in parallel (‘n_seq_max - 1‘), the function automatically processes them in sequential batches while preserving the original ordering of results.

## Value

Character vector of generated texts

---

# get_lib_path *Get Backend Library Path*

---

## Description

Returns the full path to the installed localLLM backend library.

## Usage

```
get_lib_path()
```

## Details

This function will throw an error if the backend library is not installed. Use `lib_is_installed` to check installation status first.

## Value

Character string containing the path to the backend library file.

## See Also

`lib_is_installed`, `install_localLLM`

## Examples

```
## Not run:
# Get the library path (only if installed)
if (lib_is_installed()) {
  lib_path <- get_lib_path()
  message("Library is at: ", lib_path)
}

## End(Not run)
```

---

`get_model_cache_dir` *Get the model cache directory*

---

## Description

Get the model cache directory

## Usage

```
get_model_cache_dir()
```

## Value

Path to the directory where models are cached

---

`hardware_profile` *Inspect detected hardware resources*

---

## Description

Returns the cached system profile recorded by localLLM when the package was attached. The probe captures approximate CPU, RAM, and GPU capacity so that safety warnings can estimate whether a model fits the device.

## Usage

```
hardware_profile(refresh = FALSE)
```

## Arguments

| | |
|---|---|
| `refresh` | When `TRUE`, forces a fresh probe instead of returning the cached profile. |

## Value

A list describing the operating system, CPU cores, total RAM (bytes), GPU information and detection timestamp.

## Examples

```
hardware_profile()
```

---

`install_localLLM` *Install localLLM Backend Library*

---

## Description

This function downloads and installs the pre-compiled C++ backend library required for the localLLM package to function.

## Usage

```
install_localLLM(force_cpu = FALSE, force_reinstall = FALSE)
```

## Arguments

| | |
|---|---|
| `force_cpu` | Logical. If `TRUE`, always download the CPU-only build even when a GPU is detected. Default `FALSE`. |
| `force_reinstall` | Logical. If `TRUE`, remove any existing installation and re-download. Useful for switching from a CPU build to a GPU build after installing a GPU driver. Default `FALSE`. |

## Details

This function downloads platform-specific pre-compiled binaries from GitHub releases. The backend library is stored in the user's data directory and loaded at runtime. Internet connection is required for the initial download.

On Windows and Linux, GPU support is auto-detected: if a Vulkan-capable GPU driver is found, the GPU-accelerated build is downloaded automatically. Use `force_cpu = TRUE` to override this and install the CPU build instead.

macOS always downloads the Metal-enabled build (both Apple Silicon and Intel).

## Value

Returns NULL invisibly. Called for side effects.

## See Also

`lib_is_installed`, `get_lib_path`

## Examples

```
## Not run:
# Standard install (auto-detects GPU)
install_localLLM()

# Force CPU build
install_localLLM(force_cpu = TRUE)

# Reinstall after adding a GPU driver
install_localLLM(force_reinstall = TRUE)

## End(Not run)
```

---

`intercoder_reliability` *Intercoder reliability for LLM annotations*

---

## Description

Intercoder reliability for LLM annotations

## Usage

```
intercoder_reliability(
  annotations,
  method = c("auto", "cohen", "krippendorff"),
  label_levels = NULL,
  sample_col = "sample_id",
  model_col = "model_id",
  label_col = "label"
)
```

## Arguments

| | |
|---|---|
| `annotations` | Output from [explore()] or a compatible data frame with at least ‘sample_id‘, ‘model_id‘, and ‘label‘ columns. |
| `method` | One of ‘"auto"‘, ‘"cohen"‘, or ‘"krippendorff"‘. The ‘"auto"‘ setting computes both pairwise Cohen’s Kappa and Krippendorff’s Alpha (when applicable). |
| `label_levels` | Optional factor levels to enforce a consistent ordering in the resulting tables. |
| `sample_col` | Column name that identifies samples when ‘annotations‘ is a user-provided data frame. |
| `model_col` | Column name for the model identifier when using a custom ‘annotations‘ data frame. |
| `label_col` | Column name containing model predictions when using a custom ‘annotations‘ data frame. |

### Value

A list containing ‘cohen‘ (data frame of pairwise kappas) and/or ‘krippendorff‘ (overall alpha statistic with per-item agreement scores).

---

## lib_is_installed *Check if Backend Library is Installed*

---

### Description

Checks whether the localLLM backend library has been downloaded and installed.

### Usage

```
lib_is_installed()
```

### Value

Logical value indicating whether the backend library is installed.

### See Also

install_localLLM, get_lib_path

### Examples

```
# Check if backend library is installed
if (lib_is_installed()) {
  message("Backend library is ready")
} else {
  message("Please run install_localLLM() first")
}
```

---

## list_cached_models *List cached models on disk*

---

### Description

Enumerates the models that have been downloaded to the local cache. This is useful when you want to reuse a previously downloaded model but no longer remember the original URL. The cache directory can be overridden with the ‘LOCALLLM_CACHE_DIR‘ environment variable or via the ‘cache_dir‘ argument.

### Usage

```
list_cached_models(cache_dir = NULL)
```

### Arguments

| | |
|---|---|
| cache_dir | Optional cache directory to inspect. Defaults to the package cache used by 'model_load()'. |

### Value

A data frame with one row per cached model and the columns 'name' (file name), 'path' (absolute path), 'size_bytes', and 'modified'. Returns an empty data frame when no models are cached.

---

## list_ollama_models *List GGUF models managed by Ollama*

---

### Description

This helper scans common Ollama installation directories for downloaded GGUF weights that can be loaded directly by the 'llama.cpp' backend. It inspects both manifest metadata (when available) and the blobs directory to return human-readable model descriptions.

### Usage

```
list_ollama_models(min_size_mb = 50, verify = TRUE)
```

### Arguments

| | |
|---|---|
| min_size_mb | Minimum size (in megabytes) for a candidate GGUF file. Defaults to 50 MB to avoid tiny placeholder layers. |
| verify | Whether to confirm the GGUF magic header before listing the model (default 'TRUE'). |

### Value

A data.frame with columns: 'name', 'path', 'size_mb', 'size_gb', 'size_bytes', 'sha256', 'modified', 'source', 'tag', 'model'. Returns an empty data.frame if no models are found.

---

## model_load *Load Language Model with Automatic Download Support*

---

### Description

Loads a GGUF format language model from local path or URL with intelligent caching and download management. Supports various model sources including Hugging Face, Ollama repositories, and direct HTTPS URLs. Models are automatically cached to avoid repeated downloads.

## Usage

```
model_load(
  model_path,
  cache_dir = NULL,
  n_gpu_layers = 0L,
  use_mmap = TRUE,
  use_mlock = FALSE,
  show_progress = TRUE,
  force_redownload = FALSE,
  verify_integrity = TRUE,
  check_memory = TRUE,
  hf_token = NULL,
  verbosity = 1L
)
```

## Arguments

| | |
|---|---|
| model_path | Path to local GGUF model file, URL, or cached model name. Supported URL formats:<br>• https:// - Direct download from web servers<br>If you previously downloaded a model through this package you can supply the cached file name (or a distinctive fragment of it) instead of the full path or URL. The loader will search the local cache and offer any matches. |
| cache_dir | Custom directory for downloaded models (default: NULL uses system cache directory) |
| n_gpu_layers | Number of transformer layers to offload to GPU (default: 0 for CPU-only). Set to -1 to offload all layers, or a positive integer for partial offloading |
| use_mmap | Enable memory mapping for efficient model loading (default: TRUE). Disable only if experiencing memory issues |
| use_mlock | Lock model in physical memory to prevent swapping (default: FALSE). Enable for better performance but requires sufficient RAM |
| show_progress | Display download progress bar for remote models (default: TRUE) |
| force_redownload | Force re-download even if cached version exists (default: FALSE). Useful for updating to newer model versions |
| verify_integrity | Verify file integrity using checksums when available (default: TRUE) |
| check_memory | Check if sufficient system memory is available before loading (default: TRUE). When memory is insufficient, a warning() is issued first (describing the size mismatch), followed by a stop() that aborts loading. If model_load() is wrapped in tryCatch(warning = ...), the warning handler intercepts execution before stop() runs: loading is still aborted, but the error = handler will not fire. To reliably catch the cancellation, use tryCatch(error = ...) rather than a warning = handler. |
| hf_token | Optional Hugging Face access token to set during model resolution. Defaults to the existing 'HF_TOKEN' environment variable. |

verbosity — Control backend logging during model loading (default: 1L). Larger numbers print more detail: 0 shows only errors, 1 adds warnings, 2 prints informational messages, and 3 enables the most verbose debug output.

## Value

A model object (external pointer) that can be used with context_create, tokenize, and other model functions

## See Also

context_create, download_model, get_model_cache_dir, list_cached_models

## Examples

```
## Not run:
# Load local GGUF model
model <- model_load("/path/to/my_model.gguf")

# Download from Hugging Face and cache locally
hf_path <- paste0("https://huggingface.co/Qwen/",
                  "Qwen3-0.6B-GGUF/resolve/main/Qwen3-0.6B-Q8_0.gguf")
model <- model_load(hf_path)

# Load with GPU acceleration (offload 10 layers)
model <- model_load("/path/to/model.gguf", n_gpu_layers = 10)

# Download to custom cache directory
model <- model_load(hf_path,
                    cache_dir = file.path(tempdir(), "my_models"))

# Force fresh download (ignore cache)
model <- model_load(hf_path,
                    force_redownload = TRUE)

# High-performance settings for large models
model <- model_load("/path/to/large_model.gguf",
                    n_gpu_layers = -1,     # All layers on GPU
                    use_mlock = TRUE)      # Lock in memory

# Load with minimal verbosity (quiet mode)
model <- model_load("/path/to/model.gguf", verbosity = 2L)

## End(Not run)
```

---

# model_metadata *Get All GGUF Metadata from a Loaded Model*

---

## Description

Returns all key-value metadata pairs embedded in the model's GGUF file, including architecture, tokenizer settings, chat template, and more. Useful for diagnosing issues such as missing chat templates.

## Usage

```
model_metadata(model)
```

## Arguments

| | |
|---|---|
| model | A model object from model_load |

## Value

Named character vector where names are metadata keys and values are the corresponding values. Common keys include general.architecture and tokenizer.chat_template.

## Examples

```
## Not run:
model <- model_load("model.gguf")
meta <- model_metadata(model)
# Check if chat template is embedded
meta["tokenizer.chat_template"]
# Check model architecture
meta["general.architecture"]

## End(Not run)
```

---

# quick_llama *Quick LLaMA Inference*

---

## Description

A high-level convenience function that provides one-line LLM inference. Automatically handles model downloading, loading, and text generation with optional chat template formatting and system prompts for instruction-tuned models.

## Usage

```
quick_llama(
  prompt,
  model_path = .get_default_model(),
  n_threads = NULL,
  n_gpu_layers = "auto",
  n_ctx = 2048L,
  verbosity = 1L,
```

```
  max_tokens = 100L,
  top_k = 40L,
  top_p = 1,
  temperature = 0,
  repeat_last_n = 0L,
  penalty_repeat = 1,
  system_prompt = "You are a helpful assistant.",
  auto_format = TRUE,
  chat_template = NULL,
  seed = 1234L,
  progress = interactive(),
  clean = FALSE,
  hash = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| prompt | Character string or vector of prompts to process |
| model_path | Model URL or path (default: Llama 3.2 3B Instruct Q5_K_M) |
| n_threads | Number of threads (default: auto-detect) |
| n_gpu_layers | Number of GPU layers (default: auto-detect) |
| n_ctx | Context size (default: 2048) |
| verbosity | Backend logging verbosity (default: 1L). Higher values show more detail: 0 prints only errors, 1 adds warnings, 2 includes informational messages, and 3 enables the most verbose debug output. |
| max_tokens | Maximum tokens to generate (default: 100) |
| top_k | Top-k sampling (default: 40). Limits vocabulary to k most likely tokens |
| top_p | Top-p sampling (default: 1.0). Set to 0.9 for nucleus sampling |
| temperature | Sampling temperature (default: 0.0). Higher values increase creativity |
| repeat_last_n | Number of recent tokens to consider for repetition penalty (default: 0). Set to 0 to disable |
| penalty_repeat | Repetition penalty strength (default: 1.0). Set to 1.0 to disable |
| system_prompt | System prompt to add to conversation (default: "You are a helpful assistant.") |
| auto_format | Whether to automatically apply chat template formatting (default: TRUE) |
| chat_template | Custom chat template to use (default: NULL uses model's built-in template) |
| seed | Random seed for reproducibility (default: 1234) |
| progress | Show a console progress bar during parallel generation. Defaults to interactive(). Has no effect for single-prompt runs. |
| clean | Whether to strip chat-template control tokens from the generated output. Defaults to FALSE. |
| hash | When ‘TRUE‘ (default), compute SHA-256 hashes for the prompts fed into the backend and the corresponding outputs. Hashes are attached via the ‘"hashes"‘ attribute for later inspection. |
| ... | Additional parameters passed to generate() or generate_parallel() |

## Value

Character string (single prompt) or named list (multiple prompts)

## See Also

model_load, generate, generate_parallel, install_localLLM

## Examples

```
## Not run:
# Simple usage with default settings (deterministic)
response <- quick_llama("Hello, how are you?")

# Raw text generation without chat template
raw_response <- quick_llama("Complete this: The capital of France is",
                            auto_format = FALSE)

# Custom system prompt
code_response <- quick_llama(
  "Write a Python hello world program",
  system_prompt = "You are a Python programming expert.")

# Creative writing with higher temperature
creative_response <- quick_llama("Tell me a story",
                                 temperature = 0.8,
                                 max_tokens = 200)

# Prevent repetition
no_repeat <- quick_llama("Explain AI",
                         repeat_last_n = 64,
                         penalty_repeat = 1.1)

# Multiple prompts (parallel processing)
responses <- quick_llama(c("Summarize AI", "Explain quantum computing"),
                         max_tokens = 150)

## End(Not run)
```

---

# quick_llama_reset *Reset quick_llama state*

---

## Description

Resets the cached model and context objects. The next call to ‘quick_llama()‘ will reinitialize from scratch.

## Usage

```
quick_llama_reset()
```

## Value

No return value, called for side effects (resets cached state).

---

`set_hf_token` *Configure Hugging Face access token*

---

## Description

Utility helper to manage the ‘HF_TOKEN‘ environment variable used for authenticated downloads from Hugging Face. The token is set for the current R session, and it can optionally be persisted to a ‘.Renviron‘ file for future sessions. The token is not printed back to the console.

## Usage

```
set_hf_token(token, persist = FALSE, renviron_path = NULL)
```

## Arguments

| | |
|---|---|
| `token` | Character scalar. Your Hugging Face access token, typically starting with ‘hf_‘. |
| `persist` | Logical flag controlling whether to persist the token to a startup file. Defaults to ‘FALSE‘. |
| `renviron_path` | Optional path to the ‘.Renviron‘ file to update when ‘persist = TRUE‘. Must be supplied explicitly when persisting. |

## Value

Invisibly returns the currently active token value.

## Examples

```
## Not run:
set_hf_token("hf_xxx")
tmp_env <- file.path(tempdir(), ".Renviron_localLLM")
set_hf_token("hf_xxx", persist = TRUE, renviron_path = tmp_env)

## End(Not run)
```

## smart_chat_template *Smart Chat Template Application*

### Description

Automatically detects the model type and applies the appropriate chat template. For Gemma models, uses the Gemma-specific format. For other models, falls back to 'apply_chat_template'.

### Usage

```
smart_chat_template(model, messages, template = NULL,
  add_assistant = TRUE)
```

### Arguments

| | |
|---|---|
| model | A model object created with model_load |
| messages | A list of message objects |
| template | Custom template (passed to apply_chat_template if not Gemma) |
| add_assistant | Whether to add assistant turn prefix |

### Value

Formatted chat template string

## tokenize *Convert Text to Token IDs*

### Description

Converts text into a sequence of integer token IDs that the language model can process. This is the first step in text generation, as models work with tokens rather than raw text. Different models may use different tokenization schemes (BPE, SentencePiece, etc.).

### Usage

```
tokenize(model, text, add_special = TRUE)
```

### Arguments

| | |
|---|---|
| model | A model object created with model_load |
| text | Character string or vector to tokenize. Can be a single text or multiple texts |
| add_special | Whether to add special tokens like BOS (Beginning of Sequence) and EOS (End of Sequence) tokens (default: TRUE). These tokens help models understand text boundaries |

## Value

Integer vector of token IDs corresponding to the input text. These can be used with `generate` for text generation or `detokenize` to convert back to text

## See Also

`detokenize`, `generate`, `model_load`

## Examples

```
## Not run:
# Load model
model <- model_load("path/to/model.gguf")

# Basic tokenization
tokens <- tokenize(model, "Hello, world!")
print(tokens)  # e.g., c(15339, 11, 1917, 0)

# Tokenize without special tokens (for model inputs)
raw_tokens <- tokenize(model, "Continue this text", add_special = FALSE)

# Tokenize multiple texts
batch_tokens <- tokenize(model, c("First text", "Second text"))

# Check tokenization of specific phrases
question_tokens <- tokenize(model, "What is AI?")
print(length(question_tokens))  # Number of tokens

## End(Not run)
```

---

# tokenize_test *Test tokenize function (debugging)*

---

## Description

Test tokenize function (debugging)

## Usage

```
tokenize_test(model)
```

## Arguments

| | |
|---|---|
| model | A model object |

## Value

Integer vector of tokens for "H"

validate *Validate model predictions against gold labels and peer agreement*

## Description

‘validate()‘ is a convenience wrapper that runs both [compute_confusion_matrices()] and [intercoder_reliability()] so that a single call yields per-model confusion matrices (vs gold labels and pairwise) as well as Cohen’s Kappa / Krippendorff’s Alpha scores.

## Usage

```
validate(
  annotations,
  gold = NULL,
  pairwise = TRUE,
  label_levels = NULL,
  sample_col = "sample_id",
  model_col = "model_id",
  label_col = "label",
  truth_col = "truth",
  method = c("auto", "cohen", "krippendorff"),
  include_confusion = TRUE,
  include_reliability = TRUE
)
```

## Arguments

| | |
|---|---|
| annotations | Output from [explore()] or a compatible data frame with at least ‘sample_id‘, ‘model_id‘, and ‘label‘ columns. |
| gold | Optional vector of gold labels. Overrides the ‘truth‘ column when supplied. |
| pairwise | When ‘TRUE‘, cross-model confusion tables are returned even if no gold labels exist. |
| label_levels | Optional factor levels to enforce a consistent ordering in the resulting tables. |
| sample_col, model_col, label_col, truth_col | Column names to use when ‘annotations‘ is a custom data frame. |
| method | One of ‘"auto"‘, ‘"cohen"‘, or ‘"krippendorff"‘. The ‘"auto"‘ setting computes both pairwise Cohen’s Kappa and Krippendorff’s Alpha (when applicable). |
| include_confusion | When ‘TRUE‘ (default) the confusion matrices section is included in the output. |
| include_reliability | When ‘TRUE‘ (default) the intercoder reliability section is included in the output. |

## Value

A list containing up to two elements: ‘confusion‘ (the full result of [compute_confusion_matrices()]) and ‘reliability‘ (the result of [intercoder_reliability()]). Elements are omitted when the corresponding ‘include_*‘ argument is ‘FALSE‘.

## Examples

```
annotations <- data.frame(
  sample_id = rep(1:3, times = 2),
  model_id = rep(c("llama", "qwen"), each = 3),
  label = c("pos", "neg", "pos", "pos", "neg", "neg"),
  truth = c("pos", "neg", "pos", "pos", "pos", "neg"),
  stringsAsFactors = FALSE
)

result <- validate(annotations)
names(result)
```

# Index