Package 'edgemodelr'

May 26, 2026

Type Package

Title Local Large Language Model Inference Engine

Version 0.4.1

Description Enables R users to run large language models locally using 'GGUF' model files and the 'llama.cpp' inference engine. Provides a complete R interface for loading models, generating text completions, extracting vector representations, and streaming responses in real-time.
Includes grammar-constrained generation for structured output, text classification, and retrieval-augmented generation (RAG) pipelines. Supports local inference without requiring cloud APIs or internet connectivity, ensuring complete data privacy and control. Based on the 'llama.cpp' project by Georgi Gerganov (2023) <https://github.com/ggml-org/llama.cpp>.

License MIT + file LICENSE

URL https://github.com/PawanRamaMali/edgemodelr

BugReports https://github.com/PawanRamaMali/edgemodelr/issues

Encoding UTF-8

Depends R (>= 4.0)

LinkingTo Rcpp

Imports Rcpp (>= 1.0.0), utils, tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, curl, shiny, openssl, jsonlite, httr, plumber

SystemRequirements GNU make or equivalent for building

Note Package includes self-contained 'llama.cpp' implementation (~56MB) for complete functionality without external dependencies.

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation yes

Author Pawan Rama Mali [aut, cre, cph],
Georgi Gerganov [aut, cph] (Author of llama.cpp and GGML library),
The ggml authors [cph] (llama.cpp and GGML contributors),
Jeffrey Quesnelle [ctb, cph] (YaRN RoPE implementation),
Bowen Peng [ctb, cph] (YaRN RoPE implementation),
pi6am [ctb] (DRY sampler from Koboldcpp),
Ivan Yurchenko [ctb] (Z-algorithm implementation),
Dirk Eddelbuettel [ctb, rev]

Maintainer Pawan Rama Mali <prm@outlook.in>

Repository CRAN

Date/Publication 2026-05-26 10:50:02 UTC

Contents

`build_chat_prompt` *Build chat prompt from conversation history*

Description

Build a formatted chat prompt from conversation history. When a model context is provided, uses the model's native chat template (e.g., ChatML for Qwen, Llama format for Llama models) for best results.

Usage

```
build_chat_prompt(history, ctx = NULL)
```

Arguments

`history`	List of conversation turns, each with `$role` and `$content`
`ctx`	Optional model context from `edge_load_model()`. If provided, uses the model's native chat template from GGUF metadata. If NULL, uses generic ChatML format.

Value

Formatted prompt string suitable for passing to `edge_completion()`

`edge_ask` *Ask a question using retrieval-augmented generation*

Description

Combines semantic search with text generation: retrieves relevant context from the index, then generates an answer grounded in that context. This is the core RAG (Retrieval-Augmented Generation) function.

Usage

```
edge_ask(
  ctx,
  question,
  index,
  top_k = 3L,
  n_predict = 256L,
  temperature = 0.3,
  system_prompt = NULL,
  return_context = FALSE
)
```

Arguments

ctx	Model context from edge_load_model()
question	The question to answer
index	An edge_index object from edge_index_documents
top_k	Number of context chunks to retrieve (default: 3)
n_predict	Maximum tokens for the answer (default: 256)
temperature	Sampling temperature (default: 0.3)
system_prompt	Optional system-level instruction for the model
return_context	If TRUE, return both answer and retrieved context (default: FALSE)

Value

If return_context = FALSE: a character string with the answer. If return_context = TRUE: a list with answer, context, and prompt.

See Also

edge_index_documents, edge_search

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
index <- edge_index_documents("./reports/", ctx)

answer <- edge_ask(ctx, "What were Q3 revenues?", index)
cat(answer)

# With context for debugging
result <- edge_ask(ctx, "What were Q3 revenues?", index,
  return_context = TRUE)
cat(result$answer)
print(result$context)

## End(Not run)
```

edge_benchmark *Performance benchmarking for model inference*

Description

Test inference speed and throughput with the current model to measure the effectiveness of optimizations.

Usage

```
edge_benchmark(
  ctx,
  prompt = "The quick brown fox",
  n_predict = 50,
  iterations = 3,
  track_memory = FALSE
)
```

Arguments

ctx	Model context from edge_load_model()
prompt	Test prompt to use for benchmarking (default: standard test)
n_predict	Number of tokens to generate for the test
iterations	Number of test iterations to average results
track_memory	If TRUE, attempt to report peak memory usage (best-effort)

Value

List with performance metrics

Examples

```
## Not run:
setup <- edge_quick_setup("TinyLlama-1.1B")
if (!is.null(setup$context)) {
  ctx <- setup$context
  perf <- edge_benchmark(ctx)
  print(perf)
  edge_free_model(ctx)
}

## End(Not run)
```

edge_cache_info *Cache size information*

Description

Cache size information

Usage

```
edge_cache_info(cache_dir = NULL)
```

Arguments

cache_dir	Cache directory path

Value

List with total_size_mb and file_count

edge_chat_completion *Generate a chat completion using the model's native template*

Description

Formats messages using the model's built-in chat template (read from GGUF metadata), generates a completion, and returns only the assistant's response. This is the recommended way to do multi-turn chat, as it uses the correct template format for the specific model (e.g., ChatML for Qwen, Llama format for Llama models).

Usage

```
edge_chat_completion(
  ctx,
  messages,
  n_predict = 256L,
  temperature = 0.7,
  top_p = 0.95
)
```

Arguments

ctx	Model context from edge_load_model()
messages	A list of message objects, each with `role` ("system", "user", or "assistant") and `content` (character string).
n_predict	Maximum tokens to generate (default: 256)
temperature	Sampling temperature (default: 0.7)
top_p	Nucleus sampling threshold (default: 0.95)

Value

Character string containing only the assistant's response text.

See Also

build_chat_prompt, edge_completion

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")

answer <- edge_chat_completion(ctx, list(
  list(role = "system", content = "You are a helpful assistant."),
  list(role = "user", content = "What is R?")
))
cat(answer)

edge_free_model(ctx)

## End(Not run)
```

---

edge_chat_stream *Interactive chat session with streaming responses*

Description

Interactive chat session with streaming responses

Usage

```
edge_chat_stream(ctx, system_prompt = NULL, max_history = 10, n_predict = 200L,
                 temperature = 0.8, verbose = TRUE)
```

Arguments

ctx	Model context from edge_load_model()
system_prompt	Optional system prompt to set context
max_history	Maximum conversation turns to keep in context (default: 10)
n_predict	Maximum tokens per response (default: 200)
temperature	Sampling temperature (default: 0.8)
verbose	Whether to print responses to console (default: TRUE)

Value

NULL (runs interactively)

Examples

```
## Not run:
# Requires a downloaded model (not run in checks)
setup <- edge_quick_setup("TinyLlama-1.1B")
ctx <- setup$context

if (!is.null(ctx)) {
  # Start interactive chat with streaming
  edge_chat_stream(ctx,
    system_prompt = "You are a helpful R programming assistant.")

  edge_free_model(ctx)
}

## End(Not run)
```

`edge_classify` *Classify text into predefined categories*

Description

Classify text into predefined categories using grammar-constrained generation. The grammar constraint ensures the model can only output one of the provided categories.

Usage

```
edge_classify(
  ctx,
  text,
  categories,
  instruction = NULL,
  temperature = 0.1
)
```

Arguments

`ctx`	Model context from edge_load_model()
`text`	Text to classify (character string or vector for batch)
`categories`	Character vector of allowed categories
`instruction`	Optional classification instruction
`temperature`	Sampling temperature (default: 0.1)

Value

Character string (single text) or character vector (batch) with the predicted category. Output is guaranteed to be one of the specified categories.

See Also

edge_extract, edge_grammar_completion

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")

# Single classification
result <- edge_classify(ctx,
  "I love this product!",
  categories = c("positive", "negative", "neutral"))

# Batch classification
texts <- c("Great product!", "Terrible experience", "It was okay")
results <- edge_classify(ctx, texts,
  categories = c("positive", "negative", "neutral"))

edge_free_model(ctx)

## End(Not run)
```

---

edge_clean_cache *Clean up cache directory and manage storage*

Description

Remove outdated model files from the cache directory to comply with CRAN policies about actively managing cached content and keeping sizes small.

Usage

```
edge_clean_cache(
  cache_dir = NULL,
  max_age_days = getOption("edgemodelr.cache_max_age_days", 30),
  max_size_mb = getOption("edgemodelr.cache_max_size_mb", 5000),
  use_lru = TRUE,
  ask = TRUE,
  verbose = TRUE
)
```

Arguments

cache_dir	Cache directory path (default: user cache directory)
max_age_days	Maximum age of files to keep in days (default: option edgemodelr.cache_max_age_days or 30)
max_size_mb	Maximum total cache size in MB (default: option edgemodelr.cache_max_size_mb or 5000)

use_lru	If TRUE, evict least-recently-used files when size exceeds limit
ask	Whether to ask for user confirmation before deletion (only in interactive sessions)
verbose	Whether to print status messages (default: TRUE)

Value

Invisible list of deleted files

Examples

```
## Not run:
# Clean cache files older than 30 days
edge_clean_cache()

# Clean cache with custom settings
edge_clean_cache(max_age_days = 7, max_size_mb = 100)

## End(Not run)
```

---

edge_completion *Generate text completion using loaded model*

Description

Generate text completion using loaded model

Usage

```
edge_completion(
  ctx,
  prompt,
  n_predict = 128L,
  temperature = 0.8,
  top_p = 0.95,
  timeout_seconds = NULL
)
```

Arguments

ctx	Model context from edge_load_model()
prompt	Input text prompt
n_predict	Maximum tokens to generate (default: 128)
temperature	Sampling temperature (default: 0.8)
top_p	Top-p sampling parameter (default: 0.95)
timeout_seconds	Optional timeout in seconds for inference

Value

Generated text as character string

Examples

```
## Not run:
# Requires a downloaded model (not run in checks)
model_path <- "model.gguf"
if (file.exists(model_path)) {
  ctx <- edge_load_model(model_path)
  result <- edge_completion(ctx, "The capital of France is", n_predict = 50)
  cat(result)
  edge_free_model(ctx)
}

## End(Not run)
```

---

edge_cuda_info *Check whether a CUDA backend is installed and active*

Description

Reports the installation and activation status of the GPU (CUDA) backend. Use this to verify that GPU inference is available before loading a model with `n_gpu_layers > 0`.

Usage

```
edge_cuda_info()
```

Value

A named list with three elements:

installed Logical. `TRUE` if a CUDA backend DLL is present in the edgemodelr cache directory.

active Logical. `TRUE` if the CUDA backend has been loaded into the current R session (i.e. `edge_reload_cuda()` was called).

path Character. Path to the active CUDA DLL, the installed DLL (if not yet active), or `NA` if no backend is found.

See Also

`edge_install_cuda`, `edge_install_cuda_toolkit`, `edge_reload_cuda`

Examples

```
## Not run:
info <- edge_cuda_info()
if (info$active) {
  cat("GPU inference active:", info$path, "\n")
} else if (info$installed) {
  cat("CUDA backend installed but not active. Call edge_reload_cuda().\n")
} else {
  cat("No CUDA backend. Run edge_install_cuda() to enable GPU support.\n")
}

## End(Not run)
```

---

edge_download_model *Download a GGUF model from Hugging Face*

Description

Download a GGUF model from Hugging Face

Usage

```
edge_download_model(
  model_id,
  filename = NULL,
  cache_dir = NULL,
  force_download = FALSE,
  verify_checksum = TRUE,
  expected_sha256 = NULL,
  trust_first_use = FALSE,
  verbose = TRUE
)
```

Arguments

model_id	Hugging Face model identifier (e.g., "TheBloke/TinyLlama-1.1B-Chat-v1.0-GGUF"), or a friendly model name from edge_list_models() (e.g., "Qwen3-0.6B", "mistral-7b")
filename	Specific GGUF file to download. Optional when using a friendly model name.
cache_dir	Directory to store downloaded models (default: "~/.cache/edgemodelr")
force_download	Force re-download even if file exists
verify_checksum	Verify SHA-256 checksum if available (default: TRUE)
expected_sha256	Optional expected SHA-256 hash for the model file
trust_first_use	Store a local hash if no known hash exists (default: FALSE)
verbose	Whether to print download progress messages

Value

Path to the downloaded model file

Examples

```
## Not run:
# Download using friendly name (recommended)
model_path <- edge_download_model("Qwen3-0.6B")
model_path <- edge_download_model("mistral-7b")

# Download using HuggingFace repo ID
model_path <- edge_download_model(
  model_id = "TheBloke/TinyLlama-1.1B-Chat-v1.0-GGUF",
  filename = "tinyllama-1.1b-chat-v1.0.Q4_K_M.gguf"
)

## End(Not run)
```

---

edge_download_url *Download a model from a direct URL*

Description

Downloads a GGUF model file from any URL. Supports resume and validates GGUF format. This function is useful for downloading models from GPT4All CDN or other direct sources that don't require authentication.

Usage

```
edge_download_url(
  url,
  filename,
  cache_dir = NULL,
  force_download = FALSE,
  verify_checksum = TRUE,
  expected_sha256 = NULL,
  trust_first_use = FALSE,
  verbose = TRUE
)
```

Arguments

`url`	Direct download URL for the model
`filename`	Local filename to save as
`cache_dir`	Directory to store downloaded models (default: user cache directory)
`force_download`	Force re-download even if file exists

verify_checksum
: Verify SHA-256 checksum if available (default: TRUE)

expected_sha256
: Optional expected SHA-256 hash for the file

trust_first_use
: Store a local hash if no known hash exists (default: FALSE)

verbose
: Whether to print progress messages

Value

Path to the downloaded model file

Examples

```
## Not run:
# Download from GPT4All CDN (large file, not run in checks)
model_path <- edge_download_url(
  url = "https://gpt4all.io/models/gguf/mistral-7b-instruct-v0.1.Q4_0.gguf",
  filename = "mistral-7b.gguf"
)

## End(Not run)
```

---

edge_embeddings *Extract text embeddings from a model*

Description

Computes dense vector embeddings for one or more text inputs using the loaded model. These embeddings can be used for semantic search, clustering, similarity comparison, and as input to downstream models.

Usage

```
edge_embeddings(ctx, texts, normalize = TRUE)
```

Arguments

ctx
: Model context from edge_load_model()

texts
: Character vector of texts to embed

normalize
: Whether to L2-normalize the embeddings (default: TRUE). Normalized embeddings allow cosine similarity to be computed as a simple dot product.

Details

For best results, use a model designed for embeddings (e.g., nomic-embed-text). However, generative models can also produce useful embeddings from their hidden states.

The embedding dimension depends on the model architecture. Use `edge_model_n_embd()` to query the dimension.

Value

A numeric matrix with dimensions (n_texts x n_embd), where each row is the embedding vector for the corresponding input text.

See Also

edge_similarity, edge_similarity_matrix, edge_model_n_embd

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")

# Single embedding
emb <- edge_embeddings(ctx, "The cat sat on the mat")
dim(emb)  # 1 x n_embd

# Multiple embeddings
texts <- c("cats are great", "dogs are loyal", "the stock market crashed")
embs <- edge_embeddings(ctx, texts)

# Compute similarity
edge_similarity(embs[1,], embs[2,])  # high (both about pets)
edge_similarity(embs[1,], embs[3,])  # low (different topics)

edge_free_model(ctx)

## End(Not run)
```

---

edge_extract *Extract structured data from text*

Description

High-level function that combines prompt construction with grammar-constrained generation to extract structured data from text. Returns a parsed R list.

Usage

```
edge_extract(
  ctx,
  text,
  schema,
  instruction = NULL,
  n_predict = 512L,
  temperature = 0.2
)
```

Arguments

ctx	Model context from edge_load_model()
text	The input text to analyze
schema	A named list defining the extraction schema (see edge_json_grammar)
instruction	Optional instruction to guide extraction (default: auto-generated)
n_predict	Maximum tokens to generate (default: 512)
temperature	Sampling temperature (default: 0.2, very low for factual extraction)

Value

A named list with the extracted fields, or the raw JSON string if parsing fails. Requires the **jsonlite** package for automatic parsing.

See Also

edge_json_grammar, edge_grammar_completion, edge_classify

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")

result <- edge_extract(ctx,
  "I absolutely love this product! Best purchase ever.",
  schema = list(
    sentiment = c("positive", "negative", "neutral"),
    confidence = "number"
  ))

edge_free_model(ctx)

## End(Not run)
```

---

edge_extract_batch *Extract structured data from multiple texts*

Description

Batch version of edge_extract that processes a vector of texts and returns a data frame.

Usage

```
edge_extract_batch(
  ctx,
  texts,
  schema,
  instruction = NULL,
  n_predict = 512L,
  temperature = 0.2,
  progress = TRUE
)
```

Arguments

ctx	Model context from edge_load_model()
texts	Character vector of texts to process
schema	A named list defining the extraction schema (see edge_json_grammar)
instruction	Optional instruction to guide extraction
n_predict	Maximum tokens to generate per text (default: 512)
temperature	Sampling temperature (default: 0.2)
progress	Show progress messages (default: TRUE)

Value

A data frame with one row per text and columns for each schema field

See Also

edge_extract, edge_map

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
reviews <- c("Love it!", "Worst purchase ever", "It's okay")
results <- edge_extract_batch(ctx, reviews,
  schema = list(
    sentiment = c("positive", "negative", "neutral"),
    confidence = "number"
  ))
edge_free_model(ctx)

## End(Not run)
```

edge_find_gguf_models *Find and prepare GGUF models for use with edgemodelr*

Description

This function finds compatible GGUF model files from various sources including Ollama installations, custom directories, or any folder containing GGUF files. It tests each model for compatibility with edgemodelr and creates organized copies or links for easy access.

Usage

```
edge_find_gguf_models(
  source_dirs = NULL,
  target_dir = NULL,
  create_links = TRUE,
  model_pattern = NULL,
  test_compatibility = TRUE,
  min_size_mb = 50,
  verbose = TRUE
)
```

Arguments

source_dirs	Vector of directories to search for GGUF files. If NULL, automatically searches common locations including Ollama installation.
target_dir	Directory where to create links/copies of compatible models. If NULL, creates a "local_models" directory in the current working directory.
create_links	Logical. If TRUE (default), creates symbolic links to save disk space. If FALSE, copies the files (uses more disk space but more compatible).
model_pattern	Optional pattern to filter model files by name.
test_compatibility	Logical. If TRUE (default), tests each GGUF file for compatibility with edgemodelr before including it.
min_size_mb	Minimum file size in MB to consider (default: 50MB). Helps filter out config files and focus on actual models.
verbose	Logical. Whether to print detailed progress information.

Details

This function performs the following steps:

1. Searches specified directories (or auto-detects common locations)
2. Identifies GGUF format files above the minimum size threshold
3. Optionally tests each file for compatibility with edgemodelr
4. Creates organized symbolic links or copies in the target directory

5. Returns detailed information about working models

The function automatically searches these locations if no source_dirs specified:

- Ollama models directory (~/.ollama/models or %USERPROFILE%/.ollama/models)
- Current working directory
- ~/models directory (if exists)
- Common model storage locations

Value

List containing information about compatible models, including paths and metadata

Examples

```
## Not run:
# Basic usage - auto-detect and test all GGUF models
models_info <- edge_find_gguf_models()
if (!is.null(models_info) && length(models_info$models) > 0) {
  # Load the first compatible model
  ctx <- edge_load_model(models_info$models[[1]]$path)
  result <- edge_completion(ctx, "Hello", n_predict = 20)
  edge_free_model(ctx)
}

# Search specific directories
models_info <- edge_find_gguf_models(source_dirs = c("~/Downloads", "~/models"))

# Skip compatibility testing (faster but less reliable)
models_info <- edge_find_gguf_models(test_compatibility = FALSE)

# Copy files instead of creating links
models_info <- edge_find_gguf_models(create_links = FALSE)

# Filter for specific models
models_info <- edge_find_gguf_models(model_pattern = "llama")

## End(Not run)
```

---

edge_find_ollama_models

Find and load Ollama models

Description

Utility functions to discover and work with locally stored Ollama models. Ollama stores models as SHA-256 named blobs which are GGUF files that can be used directly with edgemodelr.

Usage

```
edge_find_ollama_models(
  ollama_dir = NULL,
  test_compatibility = FALSE,
  max_size_gb = 10
)
```

Arguments

ollama_dir	Optional path to Ollama models directory. If NULL, will auto-detect.
test_compatibility	If TRUE, test if each model can be loaded successfully
max_size_gb	Maximum model size in GB to consider (default: 10)

Value

List with ollama_path and discovered models information

Examples

```
## Not run:
# Find Ollama models
ollama_info <- edge_find_ollama_models()

if (!is.null(ollama_info) && length(ollama_info$models) > 0) {
  # Use first compatible model
  model_path <- ollama_info$models[[1]]$path
  ctx <- edge_load_model(model_path)
  result <- edge_completion(ctx, "Hello", n_predict = 10)
  edge_free_model(ctx)
}

## End(Not run)
```

---

edge_free_model *Free model context and release memory*

Description

Free model context and release memory

Usage

```
edge_free_model(ctx)
```

Arguments

ctx	Model context from edge_load_model()

Value

NULL (invisibly)

Examples

```
## Not run:
# Requires a downloaded model (not run in checks)
model_path <- "model.gguf"
if (file.exists(model_path)) {
  ctx <- edge_load_model(model_path)
  # ... use model ...
  edge_free_model(ctx)  # Clean up
}

## End(Not run)
```

`edge_grammar_completion` *Generate text constrained by a GBNF grammar*

Description

Uses llama.cpp's grammar-constrained sampling to force output to conform to a GBNF grammar specification. This ensures structured, parseable output.

Usage

```
edge_grammar_completion(
  ctx,
  prompt,
  grammar,
  grammar_root = "root",
  n_predict = 512L,
  temperature = 0.3,
  top_p = 0.95
)
```

Arguments

`ctx`	Model context from edge_load_model()
`prompt`	The input prompt
`grammar`	A GBNF grammar string defining allowed output structure
`grammar_root`	The root rule name in the grammar (default: "root")
`n_predict`	Maximum tokens to generate (default: 512)
`temperature`	Sampling temperature (default: 0.3, lower for structured output)
`top_p`	Nucleus sampling threshold (default: 0.95)

Details

GBNF (GGML BNF) is a format for defining formal grammars that constrain model output. This is useful for generating JSON, XML, or any structured format.

Common GBNF patterns:

- JSON object: Use `edge_json_grammar()` for convenience
- Enum/choices: `'root ::= "yes" | "no" | "maybe"'`
- Number: `'root ::= [0-9]+'`

Value

Character string containing only the generated text (not the prompt)

See Also

edge_json_grammar, edge_extract, edge_classify

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")

# Force yes/no output
grammar <- 'root ::= "yes" | "no"'
result <- edge_grammar_completion(ctx, "Is the sky blue? Answer:", grammar)

# Force JSON output
json_grammar <- edge_json_grammar(list(
  sentiment = c("positive", "negative", "neutral"),
  confidence = "number"
))
result <- edge_grammar_completion(ctx,
  "Analyze sentiment: 'I love this product'\nJSON:",
  json_grammar)

edge_free_model(ctx)

## End(Not run)
```

---

edge_index_documents *Build an embedding index from text documents*

Description

Reads text files from a directory (or accepts text directly), splits into chunks, computes embeddings, and returns an index object for semantic search and retrieval-augmented generation.

Usage

```
edge_index_documents(
  source,
  ctx,
  chunk_size = 500L,
  chunk_overlap = 50L,
  file_pattern = "*.txt",
  normalize = TRUE,
  progress = TRUE
)
```

Arguments

source	Either a directory path containing text files, or a character vector of text chunks to index directly.
ctx	Model context from edge_load_model() (used for embedding)
chunk_size	Approximate number of characters per chunk (default: 500)
chunk_overlap	Number of characters of overlap between chunks (default: 50)
file_pattern	Glob pattern for files to read (default: "*.txt")
normalize	Normalize embeddings (default: TRUE)
progress	Show progress messages (default: TRUE)

Value

An edge_index object containing chunks, embeddings, and source metadata.

See Also

edge_search, edge_ask

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")

# Index a directory
index <- edge_index_documents("./reports/", ctx)

# Or index text directly
index <- edge_index_documents(
  c("Revenue grew 15%", "New product launched"), ctx)

# Search the index
results <- edge_search(index, ctx, "revenue growth")

## End(Not run)
```

edge_install_cuda *Install the CUDA backend for GPU-accelerated inference*

Description

Downloads a pre-built GGML CUDA backend shared library and stores it in the edgemodelr cache. After installation, restart your R session (or call `edge_reload_cuda()`) so the GPU backend is picked up during model loading.

On Windows, the CUDA runtime libraries (`cudart`, `cublas`) must also be present. Run `edge_install_cuda_toolkit()` once after this function to install them automatically.

Usage

```
edge_install_cuda(cuda_version = "13.1", force = FALSE, llama_build = "b8179")
```

Arguments

`cuda_version`	CUDA version string. One of `"12.4"` or `"13.1"` (default). Use `"13.1"` for Blackwell GPUs (RTX 50 series, sm_120).
`force`	Re-download even if already installed.
`llama_build`	llama.cpp build number to pull the backend from (default: `"b8179"`, the version bundled in this package).

Value

Invisibly returns the path to the installed CUDA backend DLL.

See Also

`edge_install_cuda_toolkit`, `edge_reload_cuda`, `edge_cuda_info`

Examples

```
## Not run:
edge_install_cuda()                  # GPU backend installed
edge_install_cuda_toolkit()          # Install CUDA runtime DLLs (Windows)
ctx <- edge_load_model("model.gguf", n_gpu_layers = 35)

## End(Not run)
```

edge_install_cuda_toolkit *Install CUDA runtime libraries required for GPU inference*

Description

The CUDA backend (`ggml-cuda-XX.dll`) requires two sets of runtime DLLs that are **not** included with the NVIDIA display driver:

- **nvcudart_hybrid64.dll** — CUDA hybrid runtime, already present on your system in the Windows DriverStore (installed with the GPU driver). This function copies it to the edgemodelr cache.
- **cublas64_13.dll / cublasLt64_13.dll** — cuBLAS linear-algebra library (~400 MB download from NVIDIA's official redistrib server).

After running this function, call `edge_reload_cuda()` and load a model with `n_gpu_layers = -1` to run on your GPU.

This function is only needed on Windows. On Linux and macOS, CUDA support is enabled at build time via the `EDGEMODELR_CUDA=1` environment variable.

Usage

```
edge_install_cuda_toolkit(cuda_version = "13.1", force = FALSE)
```

Arguments

`cuda_version`	CUDA major version string, e.g. `"13.1"` (default). Must match the version used with `edge_install_cuda()`.
`force`	Reinstall even if the runtime DLLs are already present.

Value

Invisibly returns the cuda cache directory path.

See Also

`edge_install_cuda`, `edge_reload_cuda`, `edge_cuda_info`

Examples

```
## Not run:
edge_install_cuda()          # install ggml-cuda GPU backend DLL (~140 MB)
edge_install_cuda_toolkit()  # install CUDA runtime DLLs (~400 MB, one-time)
edge_reload_cuda()           # activate in this R session
ctx <- edge_load_model("model.gguf", n_gpu_layers = -1)
result <- edge_completion(ctx, "Hello", n_predict = 20)

## End(Not run)
```

edge_json_grammar *Generate a GBNF grammar for JSON output from a schema*

Description

Converts a simple R list schema into a GBNF grammar string that constrains model output to valid JSON matching the schema.

Usage

```
edge_json_grammar(schema)
```

Arguments

schema	A named list where each element defines a field. Values can be: • `"string"` - any string value • `"number"` - a numeric value (integer or decimal) • `"integer"` - an integer value • `"boolean"` - true or false • A character vector - enum of allowed values

Value

A GBNF grammar string suitable for use with `edge_grammar_completion`

See Also

`edge_grammar_completion`, `edge_extract`

Examples

```
# Schema with enum and free-text fields
grammar <- edge_json_grammar(list(
  sentiment = c("positive", "negative", "neutral"),
  confidence = "number",
  explanation = "string"
))
cat(grammar)
```

edge_list_models *List popular pre-configured models*

Description

List popular pre-configured models

Usage

```
edge_list_models()
```

Value

Data frame with model information

edge_load_model *Load a local GGUF model for inference*

Description

Load a local GGUF model for inference

Usage

```
edge_load_model(model_path, n_ctx = 2048L, n_gpu_layers = 0L,
  n_threads = NULL, flash_attn = TRUE, embeddings = FALSE)
```

Arguments

model_path	Path to a .gguf model file
n_ctx	Maximum context length (default: 2048)
n_gpu_layers	Number of layers to offload to GPU (default: 0, CPU-only)
n_threads	Number of CPU threads for inference (default: NULL = use all hardware threads). Set to a lower value to leave cores free for other tasks.
flash_attn	Enable flash attention for faster inference (default: TRUE). Reduces memory usage and improves speed. Set to FALSE for maximum compatibility.
embeddings	Enable embedding extraction mode (default: FALSE). Must be TRUE to use edge_embeddings with this context. A model loaded with embeddings = TRUE can still be used for text generation.

Value

External pointer to the loaded model context

Examples

```
## Not run:
# Load a TinyLlama model (requires model file)
model_path <- "~/models/TinyLlama-1.1B-Chat.Q4_K_M.gguf"
if (file.exists(model_path)) {
  ctx <- edge_load_model(model_path, n_ctx = 2048)

  # Generate completion
  result <- edge_completion(ctx, "Explain R data.frame:", n_predict = 100)
  cat(result)

  # Load with threading control
  ctx2 <- edge_load_model(model_path, n_threads = 4, flash_attn = TRUE)

  # Free model when done
  edge_free_model(ctx)
}

## End(Not run)
```

`edge_load_ollama_model` *Load an Ollama model by partial SHA-256 hash*

Description

Find and load an Ollama model using a partial SHA-256 hash instead of the full path. This is more convenient than typing out the full blob path.

Usage

```
edge_load_ollama_model(partial_hash, n_ctx = 2048L, n_gpu_layers = 0L)
```

Arguments

`partial_hash`	First few characters of the SHA-256 hash
`n_ctx`	Maximum context length (default: 2048)
`n_gpu_layers`	Number of layers to offload to GPU (default: 0)

Value

Model context if successful, throws error if not found or incompatible

Examples

```
## Not run:
# Load model using first 8 characters of SHA hash
# ctx <- edge_load_ollama_model("b112e727")
# result <- edge_completion(ctx, "Hello", n_predict = 10)
# edge_free_model(ctx)

## End(Not run)
```

---

edge_map *Apply a prompt template to a vector of texts*

Description

Maps a prompt template over a character vector, generating completions for each element. This is the primary function for batch LLM operations on data frames.

Usage

```
edge_map(
  ctx,
  texts,
  prompt_template,
  n_predict = 128L,
  temperature = 0.7,
  top_p = 0.95,
  grammar = NULL,
  progress = TRUE
)
```

Arguments

ctx	Model context from edge_load_model()
texts	Character vector of input texts
prompt_template	A string with {text} as a placeholder, or a function that takes a single text and returns a prompt string.
n_predict	Maximum tokens to generate per text (default: 128)
temperature	Sampling temperature (default: 0.7)
top_p	Nucleus sampling threshold (default: 0.95)
grammar	Optional GBNF grammar string to constrain output
progress	Show progress messages (default: TRUE)

Value

Character vector of completions, same length as texts

See Also

edge_classify, edge_extract_batch

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
df <- data.frame(review = c("Great product!", "Terrible quality"))
df$summary <- edge_map(ctx, df$review, "Summarize in 5 words: {text}")
edge_free_model(ctx)

## End(Not run)
```

---

edge_model_n_embd *Get the embedding dimension of a loaded model*

Description

Returns the size of the embedding vectors produced by the loaded model.

Usage

```
edge_model_n_embd(ctx)
```

Arguments

ctx	Model context from edge_load_model()

Value

Integer giving the embedding vector dimension

See Also

edge_embeddings

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
edge_model_n_embd(ctx)  # e.g., 4096
edge_free_model(ctx)

## End(Not run)
```

edge_quick_setup *Quick setup for a popular model*

Description

Quick setup for a popular model

Usage

```
edge_quick_setup(
  model_name,
  cache_dir = NULL,
  verify_checksum = TRUE,
  expected_sha256 = NULL,
  trust_first_use = FALSE,
  verbose = TRUE
)
```

Arguments

model_name	Name of the model from edge_list_models()
cache_dir	Directory to store downloaded models
verify_checksum	Verify SHA-256 checksum if available (default: TRUE)
expected_sha256	Optional expected SHA-256 hash for the model file
trust_first_use	Store a local hash if no known hash exists (default: FALSE)
verbose	Whether to print setup progress messages

Value

List with model path and context (if llama.cpp is available)

Examples

```
## Not run:
# Quick setup with TinyLlama (downloads model, not run in checks)
setup <- edge_quick_setup("TinyLlama-1.1B")
ctx <- setup$context

if (!is.null(ctx)) {
  response <- edge_completion(ctx, "Hello!")
  cat("Response:", response, "\n")
  edge_free_model(ctx)
}

## End(Not run)
```

edge_reload_cuda *Activate an installed CUDA backend without restarting R*

Description

Loads the CUDA backend DLL into the current R session. Must be called before the first `edge_load_model()` call in the session; otherwise a session restart is required for GPU support to take effect.

Usage

```
edge_reload_cuda(path = NULL)
```

Arguments

`path`	Path to the CUDA DLL. Defaults to the standard install location (the cuda subdirectory of the edgemodelr cache).

Value

Invisibly returns `TRUE` if activation succeeded.

See Also

`edge_install_cuda`, `edge_install_cuda_toolkit`, `edge_cuda_info`

Examples

```
## Not run:
# After running edge_install_cuda() and edge_install_cuda_toolkit():
edge_reload_cuda()
ctx <- edge_load_model("model.gguf", n_gpu_layers = -1)

## End(Not run)
```

edge_search *Search an embedding index for relevant chunks*

Description

Finds the most similar text chunks to a query using cosine similarity over the embedding index.

Usage

```
edge_search(index, ctx, query, top_k = 5L)
```

Arguments

index	An edge_index object from edge_index_documents
ctx	Model context (same model used to build the index)
query	Query text string
top_k	Number of results to return (default: 5)

Value

A data frame with columns: chunk, score, source, index

See Also

edge_index_documents, edge_ask

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
index <- edge_index_documents("./docs/", ctx)
results <- edge_search(index, ctx, "quarterly revenue")
print(results)

## End(Not run)
```

---

edge_serve *Serve a model as a local OpenAI-compatible API*

Description

Starts a local Plumber API server that exposes the loaded model through endpoints compatible with the OpenAI API format. Requires the **plumber** package.

Usage

```
edge_serve(
  model_path,
  host = "127.0.0.1",
  port = 8080L,
  n_ctx = 2048L,
  n_gpu_layers = 0L,
  embeddings = FALSE,
  api_key = NULL
)
```

Arguments

model_path	Path to a .gguf model file
host	Host to bind to (default: "127.0.0.1" for local only)
port	Port number (default: 8080)
n_ctx	Context size (default: 2048)
n_gpu_layers	GPU layers (default: 0, use -1 for full GPU offload)
embeddings	Enable embeddings endpoint (default: FALSE)
api_key	Optional API key for authentication

Details

Endpoints served:

- POST /v1/completions – Text completion
- POST /v1/chat/completions – Chat completion
- POST /v1/embeddings – Text embeddings (if enabled)
- GET /v1/models – List loaded model
- GET /health – Health check

Examples

```
## Not run:
edge_serve("model.gguf", port = 8080)

# From curl:
# curl http://localhost:8080/v1/chat/completions \
#   -H "Content-Type: application/json" \
#   -d '{"messages": [{"role": "user", "content": "Hello!"}]}'

## End(Not run)
```

---

edge_set_verbose *Control llama.cpp logging verbosity*

Description

Enable or disable verbose output from the underlying llama.cpp library. By default, all output except errors is suppressed to comply with CRAN policies.

Usage

```
edge_set_verbose(enabled = FALSE)
```

Arguments

enabled	Logical. If TRUE, enables verbose llama.cpp output. If FALSE (default), suppresses all output except errors.

Value

Invisible NULL

Examples

```
# Enable verbose output (not recommended for normal use)
edge_set_verbose(TRUE)

# Disable verbose output (default, recommended)
edge_set_verbose(FALSE)
```

---

edge_simd_info *Query SIMD optimization status*

Description

Reports which SIMD (Single Instruction Multiple Data) features were enabled at compile time. This helps verify that the package is using CPU-optimized code paths for faster inference.

Usage

```
edge_simd_info()
```

Value

List with:

architecture CPU architecture (e.g., "x86_64", "aarch64")

compiler_features Character vector of compiler-detected SIMD features

ggml_features Character vector of GGML-level optimization flags

is_generic Logical; TRUE if compiled with generic (scalar) fallback

Examples

```
info <- edge_simd_info()
cat("Architecture:", info$architecture, "\n")
cat("SIMD features:", paste(info$compiler_features, collapse = ", "), "\n")
if (info$is_generic) {
 cat("Running in generic mode. Reinstall with EDGEMODELR_SIMD=AVX2 for better performance.\n")
}
```

edge_similarity *Compute cosine similarity between two embedding vectors*

Description

Computes the cosine similarity between two embedding vectors.

Usage

```
edge_similarity(a, b)
```

Arguments

a	Numeric vector (embedding)
b	Numeric vector (embedding)

Value

Cosine similarity score between -1 and 1

See Also

edge_embeddings, edge_similarity_matrix

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
embs <- edge_embeddings(ctx, c("happy cat", "joyful kitten", "stock market"))

edge_similarity(embs[1,], embs[2,])  # high (similar meaning)
edge_similarity(embs[1,], embs[3,])  # low (different topics)

## End(Not run)
```

edge_similarity_matrix *Compute a similarity matrix for a set of embeddings*

Description

Efficiently computes all pairwise cosine similarities between embedding vectors using normalized matrix multiplication.

Usage

```
edge_similarity_matrix(embeddings)
```

Arguments

embeddings	A numeric matrix where each row is an embedding vector (as returned by edge_embeddings)

Value

A symmetric numeric matrix of pairwise cosine similarities

See Also

edge_embeddings, edge_similarity

Examples

```
## Not run:
ctx <- edge_load_model("model.gguf")
embs <- edge_embeddings(ctx, c("cat", "kitten", "car", "automobile"))
sim_mat <- edge_similarity_matrix(embs)
# sim_mat[1,2] high (cat~kitten), sim_mat[1,3] low (cat~car)

## End(Not run)
```

---

edge_small_model_config

Get optimized configuration for small language models

Description

Returns recommended parameters for loading and using small models (1B-3B parameters) to maximize inference speed on resource-constrained devices.

Usage

```
edge_small_model_config(
  model_size_mb = NULL,
  available_ram_gb = NULL,
  target = "laptop"
)
```

Arguments

model_size_mb	Model file size in MB (if known). If NULL, uses conservative defaults.
available_ram_gb	Available system RAM in GB. If NULL, uses conservative defaults.
target	Device target: "mobile", "laptop", "desktop", or "server" (default: "laptop")

Value

List with optimized parameters for edge_load_model() and edge_completion()

Examples

```
# Get optimized config for a 700MB model on a laptop
config <- edge_small_model_config(model_size_mb = 700, available_ram_gb = 8)

# Use the config to load a model
## Not run:
model_path <- "path/to/tinyllama.gguf"
if (file.exists(model_path)) {
  ctx <- edge_load_model(
    model_path,
    n_ctx = config$n_ctx,
    n_gpu_layers = config$n_gpu_layers
  )

  result <- edge_completion(
    ctx,
    prompt = "Hello",
    n_predict = config$recommended_n_predict,
    temperature = config$recommended_temperature
  )

  edge_free_model(ctx)
}

## End(Not run)
```

---

edge_stream_completion

Stream text completion with real-time token generation

Description

Stream text completion with real-time token generation

Usage

```
edge_stream_completion(
  ctx,
  prompt,
  callback,
  n_predict = 128L,
  temperature = 0.8,
  top_p = 0.95,
  timeout_seconds = NULL
)
```

Arguments

ctx	Model context from edge_load_model()
prompt	Input text prompt
callback	Function called for each generated token. Receives list with token info.
n_predict	Maximum tokens to generate (default: 128)
temperature	Sampling temperature (default: 0.8)
top_p	Top-p sampling parameter (default: 0.95)
timeout_seconds	Optional timeout in seconds for inference

Value

List with full response and generation statistics

Examples

```
## Not run:
# Requires a downloaded model (not run in checks)
model_path <- "model.gguf"
if (file.exists(model_path)) {
  ctx <- edge_load_model(model_path)

  # Basic streaming with token display
  result <- edge_stream_completion(ctx, "Hello, how are you?",
    callback = function(data) {
      if (!data$is_final) {
        cat(data$token)
        flush.console()
      } else {
        cat("\n[Done: ", data$total_tokens, " tokens]\n")
      }
      return(TRUE)  # Continue generation
    })

  edge_free_model(ctx)
}

## End(Not run)
```

---

is_valid_model *Check if model context is valid*

Description

Check if model context is valid

Usage

```
is_valid_model(ctx)
```

Arguments

ctx	Model context to check

Value

Logical indicating if context is valid

---

test_ollama_model_compatibility *Test if an Ollama model blob can be used with edgemodelr*

Description

This function tries to load an Ollama GGUF blob with edgemodelr using a minimal configuration and then runs a very short completion. It is intended to quickly detect common incompatibilities (unsupported architectures, invalid or unsupported GGUF files, or models that cannot run inference) before you attempt to use the model in a longer session.

Usage

```
test_ollama_model_compatibility(model_path, verbose = FALSE)
```

Arguments

model_path	Path to the Ollama blob file (a GGUF file, typically named by its SHA-256 hash inside the Ollama models/blobs directory).
verbose	If TRUE, print human-readable diagnostics for models that fail the compatibility checks.

Details

A model is considered compatible if:

- `edge_load_model()` succeeds with a small context size (`n_ctx = 256`) and CPU-only execution (`n_gpu_layers = 0`),
- the resulting model context passes `is_valid_model()`,
- and a minimal call to `edge_completion()` (1 token) returns without error.

When `verbose = TRUE`, this function classifies common failure modes: unsupported model architecture, invalid GGUF file, unsupported GGUF version, or a generic error (first 80 characters reported with truncation indicator).

Value

Logical: TRUE if the model loads and can run a short completion successfully, FALSE otherwise.

Examples

```
## Not run:
# Test an individual Ollama blob
# is_ok <- test_ollama_model_compatibility("/path/to/blob", verbose = TRUE)
#
# This function is also used internally by edge_find_ollama_models()
# when test_compatibility = TRUE.

## End(Not run)
```

Index