Package 'rtemis.llm'

May 4, 2026

Title Large Language Models and Agentic AI

Version 0.8.1

Date 2026-04-28

Description Unified interface for creating LLM and Agent objects, generating responses and performing batch inference based on a type-checked and validated 'S7' backend. Features reasoning, structured output, memory management, and tool use. Supports 'Ollama' <https://docs.ollama.com/api>, 'OpenAI'-compatible <https://developers.openai.com/api/reference/overview>, and 'Anthropic'-compatible <https://platform.claude.com/docs/en/api/getting-started> endpoints.

License GPL (>= 3)

URL https://www.rtemis.org, https://docs.rtemis.org/r/llm, https://docs.rtemis.org/r/llm-api

BugReports https://github.com/rtemis-org/llm/issues

Encoding UTF-8

RoxygenNote 7.3.3

Imports cli, data.table, digest, httr2, jsonlite, rtemis.core, S7

Suggests keyring, testthat (>= 3.0.0), xml2

Config/testthat/edition 3

Depends R (>= 4.1.0)

NeedsCompilation no

Author E.D. Gennatas [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9280-3609>)

Maintainer E.D. Gennatas <gennatas@gmail.com>

Repository CRAN

Date/Publication 2026-05-04 12:10:02 UTC

Contents

`rtemis.llm-package` **rtemis.llm**: *Agentic AI for the rtemis ecosystem*

Description

rtemis.llm provides functionality to interface with Large Language Models, part of the rtemis ecosystem. It provides an `Agent` class with support for reasoning, structured output, memory management, and tool use. Allows creation of custom LLM-based workflows and agentic AI systems using a functional user-facing frontend and an S7 backend. Includes `llmapply()` for quick batch LLM inference. Supports Ollama, OpenAI, and Anthropic endpoints.

Online Documentation and Vignettes

https://www.rtemis.org

Author(s)

Maintainer: E.D. Gennatas <gennatas@gmail.com> (ORCID) [copyright holder]

See Also

Useful links:

- https://www.rtemis.org
- https://docs.rtemis.org/r/llm
- https://docs.rtemis.org/r/llm-api
- Report bugs at https://github.com/rtemis-org/llm/issues

---

agentapply *Apply an Agent over a vector of prompts*

Description

`agentapply` is the `lapply`-style entry point for running a single prompt against an `Agent` repeatedly over a vector of inputs. Pass either a model name (in which case an `Agent` is built on the fly using `backend`, `system_prompt`, `tools`, `use_memory`, `max_tool_rounds`, `output_schema`) or a pre-built `Agent` object.

Usage

```
agentapply(
  x,
  model_or_agent,
  backend = c("ollama", "openai", "anthropic"),
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  tools = NULL,
  use_memory = FALSE,
  max_tool_rounds = 3L,
  output_schema = NULL,
  verbosity = 1L,
  extract_responses = TRUE,
  ...
)
```

Arguments

x	Character or list: Values to iterate over.
model_or_agent	Character or Agent: Either the name of a model (a string) or a pre-built Agent object from create_agent.
backend	Character {"ollama", "openai", "anthropic"}: Backend to use when model_or_agent is a string. Ignored when model_or_agent is an Agent object.
system_prompt	Character: System prompt for the on-the-fly Agent.
tools	Optional list of Tool objects: Tools available to the on-the-fly Agent.
use_memory	Logical: Whether the on-the-fly Agent should keep conversation memory.
max_tool_rounds	Integer [1, Inf): Maximum number of tool call rounds per query.
output_schema	Optional Schema: Output schema for the on-the-fly Agent.
verbosity	Integer [0, Inf): Verbosity level.
extract_responses	Logical: If TRUE, return a character vector of assistant responses. If FALSE, return the raw list of lists of Message objects.
...	Additional per-call arguments forwarded to generate.

Details

Unlike llmapply, this function can carry tools and memory. The default is use_memory = FALSE because the common case for vectorized calls is independent queries.

Value

If extract_responses = TRUE, a character vector the same length as x. Otherwise, a list of lists of Message objects.

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  agentapply(
    c("today", "yesterday", "tomorrow"),
    "gemma4:e4b",
    system_prompt = "Return the date in ISO format",
    tools = list(tool_datetime),
    temperature = 0.2
  )

## End(Not run)
```

`anthropic_check_model` *Check Anthropic Model Is Available*

Description

Check Anthropic Model Is Available

Usage

```
anthropic_check_model(
  x,
  base_url = ANTHROPIC_URL_DEFAULT,
  api_key = NULL,
  api_key_env = ANTHROPIC_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  anthropic_version = ANTHROPIC_API_VERSION_DEFAULT
)
```

Arguments

`x`	Character: Name of the model.
`base_url`	Character: Base URL of the Anthropic API.
`api_key`	Optional character: API key.
`api_key_env`	Character: Environment variable containing the API key.
`keychain_service`	Optional character: macOS Keychain service containing the API key.
`anthropic_version`	Character: `anthropic-version` header value.

Value

NULL, invisibly, if the model is available; otherwise throws an error.

Author(s)

EDG

Examples

```
# Requires running Anthropic-compatible server with /models endpoint
## Not run:
  anthropic_check_model(
    x = "test-model",
    base_url = "http://localhost:1234/v1",
    api_key = "test-key"
  )

## End(Not run)
```

`anthropic_list_models` *List Anthropic (Anthropic) Models*

Description

List Anthropic (Anthropic) Models

Usage

```
anthropic_list_models(
  base_url = ANTHROPIC_URL_DEFAULT,
  api_key = NULL,
  api_key_env = ANTHROPIC_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  anthropic_version = ANTHROPIC_API_VERSION_DEFAULT
)
```

Arguments

`base_url`	Character: Base URL of the Anthropic API.
`api_key`	Optional character: API key.
`api_key_env`	Character: Environment variable containing the API key.
`keychain_service`	Optional character: macOS Keychain service containing the API key.
`anthropic_version`	Character: `anthropic-version` header value.

Value

Character vector: Model ids.

Author(s)

EDG

Examples

```
# Requires running Anthropic-compatible server with /models endpoint
## Not run:
  anthropic_list_models(
    base_url = "http://localhost:1234/v1",
   api_key = "test-key"
  )

## End(Not run)
```

as.list.Message *Convert Message to List*

Description

Convert Message to List

Usage

```
## S3 method for class 'Message'
as.list(x, ...)
```

Arguments

x	Message object
...	Additional arguments (not used)

Value

A list representation of the Message object

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
  ## Not run:
  llm <- create_Ollama("gemma4:e4b")
  res <- generate(llm, "How can anything exist?")
  as.list(res)

## End(Not run)
```

as_list *Convert to R list*

Description

Generic method to convert various objects to R lists

Usage

```
as_list(x, ...)
```

Arguments

x	An object to convert
...	Additional arguments for specific methods

Value

A named R list

Author(s)

EDG

Examples

```
decay_time <- field("decay_time", "Time from peak amplitude to sustain level", type = "number")
as_list(decay_time)
```

---

available_tools *Print built-in tools available for use by agents*

Description

Prints the R handle (`tool_*`), the `function_name` the model sees, and the description of every built-in `Tool` exported by the package. Derived at call time from the namespace — no hardcoded list.

Usage

```
available_tools(verbosity = 1L)
```

Arguments

verbosity	Integer: Verbosity level.

Value

A named list of `Tool` objects keyed by their R handle, invisibly.

Author(s)

EDG

Examples

```
available_tools()
```

config_Anthropic *Create a AnthropicConfig Object*

Description

Creates a AnthropicConfig object which can be passed to `create_agent()`

Usage

```
config_Anthropic(
  model_name,
  temperature = TEMPERATURE_DEFAULT,
  base_url = ANTHROPIC_URL_DEFAULT,
  api_key = NULL,
  api_key_env = ANTHROPIC_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  anthropic_version = ANTHROPIC_API_VERSION_DEFAULT,
  anthropic_beta = NULL,
  max_tokens = ANTHROPIC_MAX_TOKENS_DEFAULT,
  timeout = ANTHROPIC_TIMEOUT_DEFAULT,
  extra_headers = NULL,
  extra_body = NULL,
  thinking_budget_tokens = NULL,
  validate_model = FALSE
)
```

Arguments

`model_name`	Character: The name of the Anthropic model to use (for example `"claude-sonnet-4-6"`).
`temperature`	Numeric [0, 2]: The temperature for the model.
`base_url`	Character: Base URL of the Anthropic API.
`api_key`	Optional character: API key.
`api_key_env`	Character: Environment variable containing the API key.
`keychain_service`	Optional character: macOS Keychain service containing the API key.
`anthropic_version`	Character: Value of the required `anthropic-version` header.
`anthropic_beta`	Optional character: Value(s) for the `anthropic-beta` header. A character vector is comma-joined.
`max_tokens`	Integer [1, Inf): Maximum number of tokens the model may generate. Required by the Messages API.
`timeout`	Numeric (0, Inf): Request timeout in seconds.
`extra_headers`	Optional list: Additional HTTP headers.
`extra_body`	Optional list: Additional request body fields.

thinking_budget_tokens
: Optional integer [1024, Inf): Budget for extended thinking. When set, each request enables extended thinking with this budget.

validate_model
: Logical: Whether to validate model availability using `/models`.

Value

AnthropicConfig object

Author(s)

EDG

Examples

```
cfg <- config <- config_Anthropic(
   model_name = "claude-sonnet-4-6",
   temperature = 0.4,
   api_key = "test-key",
   max_tokens = 1024L,
   validate_model = FALSE
)
```

---

config_Ollama *Create an OllamaConfig Object*

Description

Creates an OllamaConfig object which can be passed to `create_agent()`

Usage

```
config_Ollama(
  model_name,
  temperature = TEMPERATURE_DEFAULT,
  base_url = OLLAMA_URL_DEFAULT,
  think = NULL
)
```

Arguments

model_name
: Character: The name of the LLM model to use. Must be an Ollama model.

temperature
: Numeric: The temperature for the model.

base_url
: Character: Base URL of Ollama server.

think
: Optional Logical or Character {"low", "medium", "high"}: Default thinking mode for this config. Logical values target models like deepseek or qwen3; character values target gpt-oss. Can be overridden per call.

Value

OllamaConfig object

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  config_Ollama(
    model_name = "gemma4:e4b",
    temperature = 0.2
  )

## End(Not run)
```

---

config_OpenAI *Create an OpenAI-compatible Config Object*

Description

Creates an OpenAIConfig object which can be passed to `create_agent()`

Usage

```
config_OpenAI(
  model_name,
  temperature = TEMPERATURE_DEFAULT,
  base_url = OPENAI_URL_DEFAULT,
  api_key = NULL,
  api_key_env = OPENAI_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  organization = NULL,
  project = NULL,
  timeout = OPENAI_TIMEOUT_DEFAULT,
  extra_headers = NULL,
  extra_body = NULL,
  enable_thinking = NULL,
  validate_model = FALSE
)
```

Arguments

model_name	Character: The name of the LLM model to use.
temperature	Numeric [0, 2]: The temperature for the model.
base_url	Character: Base URL of the OpenAI-compatible server.
api_key	Optional character: API key.
api_key_env	Character: Environment variable containing the API key.
keychain_service	Optional character: macOS Keychain service containing the API key.
organization	Optional character: OpenAI organization id.
project	Optional character: OpenAI project id.
timeout	Numeric (0, Inf): Request timeout in seconds.
extra_headers	Optional list: Additional HTTP headers.
extra_body	Optional list: Additional request body fields.
enable_thinking	Optional logical: Whether to enable model thinking for compatible local servers.
validate_model	Logical: Whether to validate model availability using the models endpoint.

Value

OpenAIConfig object

Author(s)

EDG

Examples

```
cfg <- config_OpenAI(
   model_name = "local-model",
   temperature = 0.4,
   base_url = "http://localhost:1234/v1/",
   validate_model = FALSE
)
```

---

create_agent *Create a rtemis.llm Agent*

Description

Create a rtemis.llm Agent

Usage

```
create_agent(
  llmconfig,
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  use_memory = TRUE,
  tools = NULL,
  max_tool_rounds = 3L,
  output_schema = NULL,
  name = NULL,
  allow_custom_tools = FALSE,
  logfile = NULL,
  verbosity = 1L
)
```

Arguments

`llmconfig`	`LLMConfig`: The LLM configuration to use. Create using one of config_Ollama, config_OpenAI, or config_Anthropic.
`system_prompt`	Optional character: The system prompt to use.
`use_memory`	Logical: Whether to use conversation memory.
`tools`	Optional list of Tool objects: The tools available to the agent.
`max_tool_rounds`	Integer: Maximum number of tool call rounds per query.
`output_schema`	Optional Schema: The output schema to enforce on the agent's response created using schema and field.
`name`	Optional character: The name of the agent.
`allow_custom_tools`	Logical: If TRUE, allow the agent to carry tools whose `function_name` is not in the package allowlist. Such tools must be built via create_custom_tool and supply their own function body. The caller vouches for that code: built-in package guarantees (allowlist + hash verification) do not apply to it. Defaults to FALSE.
`logfile`	Optional character: Path to the agent's security log. Important! If NULL, the value will be set to `getOption("rtemis_security_logfile", tempfile("rtemis_security_log_", fileext = ".jsonl"))` to satisfy CRAN policy. It is important to set it to a non-temporary location that will persist and you can access. Otherwise, security incidents may be missed. Can be overridden per call on generate.
`verbosity`	Integer: Verbosity level.

Value

`Agent` object

Author(s)

EDG

Examples

```
# Requires Ollama server running and gemma4:e4b model available
## Not run:
  agent <- create_agent(
    config_Ollama(
      model_name = "gemma4:e4b",
      temperature = 0.2
    ),
   system_prompt = "You are professor of Trance at the Institute of Advanced Beat Studies.",
    use_memory = TRUE
  )

## End(Not run)
```

`create_Anthropic` *Create a Anthropic LLM Object*

Description

Create a Anthropic LLM Object

Usage

```
create_Anthropic(
  model_name,
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  temperature = TEMPERATURE_DEFAULT,
  output_schema = NULL,
  name = NULL,
  base_url = ANTHROPIC_URL_DEFAULT,
  api_key = NULL,
  api_key_env = ANTHROPIC_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  anthropic_version = ANTHROPIC_API_VERSION_DEFAULT,
  anthropic_beta = NULL,
  max_tokens = ANTHROPIC_MAX_TOKENS_DEFAULT,
  timeout = ANTHROPIC_TIMEOUT_DEFAULT,
  extra_headers = NULL,
  extra_body = NULL,
  thinking_budget_tokens = NULL,
  validate_model = FALSE
)
```

Arguments

`model_name`	Character: The name of the Anthropic model to use.
`system_prompt`	Character: The system prompt to use.

temperature	Numeric [0, 2]: The temperature for the model.
output_schema	Optional Schema: Output schema created using schema. Structured output is implemented by forcing a single synthetic tool call whose input_schema is this schema.
name	Optional character: Name for the LLM object.
base_url	Character: Base URL of the Anthropic API.
api_key	Optional character: API key.
api_key_env	Character: Environment variable containing the API key.
keychain_service	Optional character: macOS Keychain service containing the API key.
anthropic_version	Character: Value of the required anthropic-version header.
anthropic_beta	Optional character: Value(s) for the anthropic-beta header.
max_tokens	Integer [1, Inf): Maximum number of tokens the model may generate.
timeout	Numeric (0, Inf): Request timeout in seconds.
extra_headers	Optional list: Additional HTTP headers.
extra_body	Optional list: Additional request body fields.
thinking_budget_tokens	Optional integer [1024, Inf): Extended-thinking budget.
validate_model	Logical: Whether to validate model availability using /models.

Value

Anthropic LLM object

Author(s)

EDG

Examples

```
llm <- create_Anthropic(
   model_name = "claude-sonnet-4-6",
   system_prompt = "You are a meticulous research assistant.",
   api_key = "test-key",
   validate_model = FALSE
)
```

create_custom_tool *create_custom_tool*

Description

Define a user-supplied tool for an agent. Unlike create_tool, the caller provides the R function to invoke (`impl`). Custom tools are outside the package's allowlist-and-hash enforcement, so the caller vouches for the code. An agent will refuse to carry a custom tool unless the agent is created with `allow_custom_tools = TRUE` (see create_agent).

Usage

```
create_custom_tool(name, function_name, description, parameters = list(), impl)
```

Arguments

`name`	Character: The name of the tool, e.g. "Addition".
`function_name`	Character: The name to expose to the model, e.g. "add_numbers".
`description`	Character: The description of the tool.
`parameters`	List of `ToolParameter`: The parameters of the tool, each defined using tool_param.
`impl`	Function: The R function to invoke when the tool is called. Its formal argument names must match the `name` fields of `parameters`.

Value

Tool object with `impl` populated.

Author(s)

EDG

Examples

```
add_numbers <- function(x, y) x + y
tool_addition <- create_custom_tool(
  name = "Addition",
  function_name = "add_numbers",
  description = "Performs arithmetic addition of two numbers.",
  parameters = list(
    tool_param("x", "number", "The first number to add", required = TRUE),
    tool_param("y", "number", "The second number to add", required = TRUE)
  ),
  impl = add_numbers
)
```

`create_Ollama` *Create an Ollama Object*

Description

Create an Ollama Object

Usage

```
create_Ollama(
  model_name,
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  temperature = TEMPERATURE_DEFAULT,
  output_schema = NULL,
  name = NULL,
  base_url = OLLAMA_URL_DEFAULT,
  think = NULL
)
```

Arguments

`model_name`	Character: The name of the LLM model to use. Must be an Ollama model.
`system_prompt`	Character: The system prompt to use.
`temperature`	Numeric: The temperature for the model.
`output_schema`	Optional Schema: An optional output schema created using schema.
`name`	Character or NULL: An optional name for the Ollama object.
`base_url`	Character: Base URL of Ollama server.
`think`	Optional Logical or Character {"low", "medium", "high"}: Default thinking mode. Logical values target models like deepseek or qwen3; character values target gpt-oss.

Value

Ollama LLM object

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  llm <- create_Ollama(
    model_name = "gemma4:e4b",
   system_prompt = "You are professor of Drum and Bass at the Institute of Advanced Beat Studies.",
```

```
  temperature = 1.0
)
generate(llm, "What is your name and who made you?")

## End(Not run)
```

`create_OpenAI` *Create an OpenAI-compatible LLM Object*

Description

Create an OpenAI-compatible LLM Object

Usage

```
create_OpenAI(
  model_name,
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  temperature = TEMPERATURE_DEFAULT,
  output_schema = NULL,
  name = NULL,
  base_url = OPENAI_URL_DEFAULT,
  api_key = NULL,
  api_key_env = OPENAI_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  organization = NULL,
  project = NULL,
  timeout = OPENAI_TIMEOUT_DEFAULT,
  extra_headers = NULL,
  extra_body = NULL,
  enable_thinking = NULL,
  validate_model = FALSE
)
```

Arguments

`model_name`	Character: The name of the LLM model to use.
`system_prompt`	Character: The system prompt to use.
`temperature`	Numeric [0, 2]: The temperature for the model.
`output_schema`	Optional Schema: Output schema created using schema.
`name`	Optional character: Name for the LLM object.
`base_url`	Character: Base URL of the OpenAI-compatible server.
`api_key`	Optional character: API key.
`api_key_env`	Character: Environment variable containing the API key.

keychain_service	Optional character: macOS Keychain service containing the API key.
organization	Optional character: OpenAI organization id.
project	Optional character: OpenAI project id.
timeout	Numeric (0, Inf): Request timeout in seconds.
extra_headers	Optional list: Additional HTTP headers.
extra_body	Optional list: Additional request body fields.
enable_thinking	Optional logical: Whether to enable model thinking for compatible local servers.
validate_model	Logical: Whether to validate model availability using the models endpoint.

Value

OpenAI LLM object

Author(s)

EDG

Examples

```
llm <- create_OpenAI(
   model_name = "local-model",
   base_url = "http://localhost:1234/v1",
   system_prompt = "You are a meticulous research assistant.",
   validate_model = FALSE
)
```

---

create_tool *create_tool*

Description

Define a tool for an agent

Usage

```
create_tool(name, function_name, description, parameters = list())
```

Arguments

name	Character: The name of the tool, e.g. "Wikipedia Search".
function_name	Character: The name of the function to call, e.g. "query_wikipedia".
description	Character: The description of the tool.
parameters	List of `ToolParameter`: The parameters of the tool, each defined using tool_param.

Value

Tool object

Author(s)

EDG

Examples

```
tool_addition <- create_tool(
  name = "Addition",
  function_name = "add_numbers",
  description = "Performs arithmetic addition of two numbers.",
  parameters = list(
    tool_param(
      name = "x",
      type = "number",
      description = "The first number to add",
      required = TRUE
    ),
    tool_param(
      name = "y",
      type = "number",
      description = "The second number to add",
      required = TRUE
    )
  )
)
```

---

field *Define a schema field*

Description

Define a schema field

Usage

```
field(
  name,
  description = name,
  type = c("string", "number", "integer", "boolean", "array", "object"),
  required = TRUE
)
```

Arguments

name	Optional Character: The name of the field.
description	Optional Character: A brief description of the field.
type	Character {"string", "number", "integer", "boolean", "array", "object"}: The field type.
required	Logical: Whether the field is required.

Value

Field object

Author(s)

EDG

Examples

```
# `type` defaults to "string", `required` defaults to TRUE
field("lab_name", "Name of the lab test")
field("normal_range_low", "Lower bound of normal range", type = "number")
```

---

generate *Generate Method*

Description

Generic method for generating text or structured output from LLMs and Agents.

Usage

```
generate(
  x,
  prompt,
  temperature = NULL,
  top_p = NULL,
  max_tokens = NULL,
  stop = NULL,
  think = NULL,
  output_schema = NULL,
  verbosity = 1L,
  ...
)
```

Arguments

x	An object of class LLM or Agent.
prompt	Character: The prompt to pass to the model or agent.
temperature	Optional numeric [0, 2]: Per-call sampling temperature.
top_p	Optional numeric [0, 1]: Nucleus sampling cutoff.
max_tokens	Optional integer [1, Inf): Maximum tokens to generate. For Anthropic, this overrides the config-level value (which is required); for Ollama this maps to options.num_predict; for OpenAI-compatible backends this maps to max_tokens.
stop	Optional character: Stop sequence(s). Mapped to stop_sequences on Anthropic and options.stop on Ollama.
think	Optional logical or character: Whether to enable model thinking (reasoning trace) for this call. Character values target gpt-oss-style local models.
output_schema	Optional Schema: Output schema to enforce on this call's response. If omitted, the object's default schema (if any) is used.
verbosity	Integer: Verbosity level.
...	Additional backend-specific per-call arguments. See Details.

Details

The system prompt is set once at agent (or LLM) construction time and is **not** overridable per call. Construct a new agent if you need a different system prompt.

Backend-specific extra arguments accepted via ...:

- **Ollama**: top_k (integer), seed (integer)
- **OpenAI**: seed (integer)
- **Anthropic**: top_k (integer)

Any argument set to NULL (the default) falls back to the value baked into the underlying LLMConfig at construction time.

Value

Message object or list of Message objects (for Agent).

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  agent <- create_agent(
    config_Ollama(
      model_name = "gemma4:e4b",
      temperature = 0.2
```

```
  )
)
generate(agent, "What is your name?", temperature = 0.7)

## End(Not run)
```

`llmapply` *Apply an LLM over a vector of prompts*

Description

`llmapply` is the `lapply`-style entry point for running a single prompt against an `LLM` repeatedly over a vector of inputs. Pass either a model name (in which case an `LLM` is built on the fly using `backend`, `system_prompt`, `output_schema`) or a pre-built `LLM` object.

Usage

```
llmapply(
  x,
  model_or_llm,
  backend = c("ollama", "openai", "anthropic"),
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  output_schema = NULL,
  verbosity = 1L,
  extract_responses = TRUE,
  ...
)
```

Arguments

`x`	Character or list: Values to iterate over. Each element forms the user prompt for one call to the LLM.
`model_or_llm`	Character or LLM: Either the name of a model (a string) or a pre-built `LLM` object (for example from create_Ollama, create_OpenAI, or create_Anthropic).
`backend`	Character {"ollama", "openai", "anthropic"}: Backend to use when `model_or_llm` is a string. Ignored when `model_or_llm` is an `LLM` object.
`system_prompt`	Character: System prompt to use when building the `LLM` from a model name. Ignored when `model_or_llm` is an `LLM` object.
`output_schema`	Optional Schema: Output schema to enforce, created with schema. When `model_or_llm` is a string, this is baked into the built `LLM`. When `model_or_llm` is a pre-built `LLM`, supplying this here is a conflict and will error.
`verbosity`	Integer [0, Inf): Verbosity level. The per-call verbosity is `verbosity - 1L`.
`extract_responses`	Logical: If `TRUE`, return a character vector of assistant responses (with `NA_character_` for missing assistant content). If `FALSE`, return the raw list of `Message` objects from each call.
`...`	Additional per-call arguments forwarded to generate (e.g. `temperature`, `top_p`, `max_tokens`, `stop`, `think`, `top_k`, `seed`).

Details

Per-call overrides such as `temperature`, `top_p`, `max_tokens`, `stop`, `think`, plus backend-specific options like `top_k` or `seed`, are forwarded via `...` to generate. Vectors passed via `...` are **not** yet recycled across `x` — they are forwarded as-is to each call.

Value

If `extract_responses = TRUE`, a character vector the same length as `x`. Otherwise, a list of `Message` objects.

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  llmapply(
    c("burgundy", "crimson", "maroon", "ruby", "scarlet"),
    "gemma4:e4b",
   system_prompt = "Return the hexadecimal code for the color provided in format #FFFFFF",
    temperature = 0.2
  )

## End(Not run)
```

`map` *Map*

Description

Map

Usage

```
map(x, f, ...)
```

Arguments

`x`	A character vector or list to map over.
`f`	An `LLM` or `Agent` object.
`...`	Additional arguments passed to `generate()`.

Details

Use responses to retrieve just the content from the assistant messages, or reasoning to retrieve the reasoning traces (if enabled).

Value

A list of Message objects (for LLM) or list of lists of Message objects (for Agent).

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  llm <- create_Ollama(
    "gemma4:e4b",
    system_prompt = "Convert color to hex code using the format #FFFFFF"
  )
  x <- c("ocean teal", "california poppy orange", "bougainvillea pink")
  hex <- map(x, llm)
  hex

## End(Not run)
```

---

ollama_check_model *Check Ollama Model is Available*

Description

Check Ollama Model is Available

Usage

```
ollama_check_model(x)
```

Arguments

x	Character: Name of model.

Value

NULL, invisibly if model is available; otherwise throws an error.

Author(s)

EDG

Examples

```
# Requires running Ollama server
## Not run:
  ollama_check_model("gemma4:e4b")

## End(Not run)
```

`ollama_get_model_info` *Get Ollama Model Info*

Description

Get Ollama Model Info

Usage

```
ollama_get_model_info(x = NULL, base_url = OLLAMA_URL_DEFAULT)
```

Arguments

`x`	Optional character vector: Name of model(s) to get info for. If NULL, all available models' info is returned.
`base_url`	Character: Base URL of Ollama server.

Value

data.table

Author(s)

EDG

Examples

```
# Requires a running Ollama server
## Not run:
  ollama_get_model_info()
  ollama_get_model_info(x = "gemma4:e4b")

## End(Not run)
```

ollama_list_models *List Ollama Models*

Description

List Ollama Models

Usage

```
ollama_list_models(base_url = OLLAMA_URL_DEFAULT)
```

Arguments

base_url	Character: Base URL of Ollama server.

Value

Character vector: Model names.

Author(s)

EDG

Examples

```
# Requires a running Ollama server
## Not run:
ollama_list_models()

## End(Not run)
```

openai_check_model *Check OpenAI-compatible Model Is Available*

Description

Check OpenAI-compatible Model Is Available

Usage

```
openai_check_model(
  x,
  base_url = OPENAI_URL_DEFAULT,
  api_key = NULL,
  api_key_env = OPENAI_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  organization = NULL,
  project = NULL
)
```

Arguments

x	Character: Name of model.
base_url	Character: Base URL of the OpenAI-compatible server.
api_key	Optional character: API key.
api_key_env	Character: Environment variable containing the API key.
keychain_service	Optional character: macOS Keychain service containing the API key.
organization	Optional character: OpenAI organization id.
project	Optional character: OpenAI project id.

Value

NULL, invisibly, if model is available; otherwise throws an error.

Author(s)

EDG

Examples

```
# Requires running OpenAI-compatible server with /models endpoint
## Not run:
  openai_check_model(
    x = "local-model",
    base_url = "http://localhost:1234/v1",
    api_key = "test-key"
  )

## End(Not run)
```

---

openai_list_models *List OpenAI-compatible Models*

Description

List OpenAI-compatible Models

Usage

```
openai_list_models(
  base_url = OPENAI_URL_DEFAULT,
  api_key = NULL,
  api_key_env = OPENAI_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  organization = NULL,
  project = NULL
)
```

Arguments

base_url	Character: Base URL of the OpenAI-compatible server.
api_key	Optional character: API key.
api_key_env	Character: Environment variable containing the API key.
keychain_service	Optional character: macOS Keychain service containing the API key.
organization	Optional character: OpenAI organization id.
project	Optional character: OpenAI project id.

Value

Character vector: Model ids.

Author(s)

EDG

Examples

```
# Requires running OpenAI-compatible server with /models endpoint
## Not run:
  openai_list_models(
    base_url = "http://localhost:1234/v1",
    api_key = "test-key"
  )

## End(Not run)
```

---

reasoning *Extract reasoning trace(s) from a Message or list of Messages*

Description

Returns the assistant's reasoning trace (if any). Only LLMMessage objects carry a reasoning field; all other Message subclasses return NA_character_. Messages whose reasoning is unset (NULL) also return NA_character_.

Usage

```
reasoning(x)
```

Arguments

x	Message object, list of Message objects, or list of lists of Message objects.

Value

Character vector of reasoning traces, with NA_character_ in slots where no reasoning is available.

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  llmapply(
    c("burgundy", "crimson", "maroon", "ruby", "scarlet"),
    "gemma4:e4b",
   system_prompt = "Return the hexadecimal code for the color provided in format #FFFFFF",
    temperature = 0.2
  ) |> reasoning()

## End(Not run)
```

---

responses *Extract response(s) from a Message or list of Messages*

Description

Returns the assistant content from a single Message, from a flat list of Message objects (e.g. the output of llmapply with extract_responses = FALSE), or from a list of lists of Message objects (e.g. the output of map() on an Agent with extract_responses = FALSE).

Usage

```
responses(x)
```

Arguments

x	Message object, list of Message objects, or list of lists of Message objects.

Value

Character vector of assistant responses. Returns NA_character_ in slots where no assistant message is present, so that the length of the result matches the length of x.

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
  llmapply(
    c("burgundy", "crimson", "maroon", "ruby", "scarlet"),
    "gemma4:e4b",
   system_prompt = "Return the hexadecimal code for the color provided in format #FFFFFF",
    temperature = 0.2
  ) |> responses()

## End(Not run)
```

---

schema *Define output schema for LLM responses*

Description

Define output schema for LLM responses

Usage

```
schema(name = NULL, ..., description = NULL)
```

Arguments

name	Optional Character: The name of the schema.
...	Field objects defining the schema fields. Create using field.
description	Optional Character: A brief description of the schema.

Value

Schema object, named list, or JSON string.

Author(s)

EDG

Examples

```
schema(
  "LabSchema",
  field("Lab name"),
  field("normal range low", type = "number"),
  field("normal range high", type = "number")
)
```

tools *Built-in Agent Tools*

Description

Pre-defined `Tool` objects that can be passed to `create_agent()` via the `tools` argument, allowing an agent to search external services or retrieve local information.

Usage

```
tool_arxiv

tool_datetime

tool_duckduckgo_ia

tool_semanticscholar

tool_wikipedia
```

Format

`Tool` S7 objects:

`tool_arxiv` Search arXiv.org for academic papers.

`tool_wikipedia` Search Wikipedia articles.

`tool_semanticscholar` Search Semantic Scholar for academic papers.

`tool_duckduckgo_ia` Query the DuckDuckGo Instant Answer API.

`tool_datetime` Return the current date, time, and timezone.

Author(s)

EDG

Examples

```
# Inspect a tool
tool_datetime

## Not run:
# Requires a running Ollama server and the "gemma4:e4b" model
agent <- create_agent(
  llmconfig = config_Ollama(
    model_name = "gemma4:e4b",
    base_url = "http://localhost:11434"
  ),
  system_prompt = "You are a meticulous research assistant.",
```

```
  tools = list(tool_datetime, tool_semanticscholar, tool_wikipedia)
)
generate(agent, "Find recent papers on diffusion models.")

## End(Not run)
```

`tool_param` *tool_param*

Description

Define a tool parameter schema

Usage

```
tool_param(name, type, description, required = FALSE)
```

Arguments

`name`	Character: The name of the parameter.
`type`	Character: The type of the parameter.
`description`	Character: The description of the parameter.
`required`	Logical: Whether the parameter is required.

Value

ToolParameter object

Author(s)

EDG

Examples

```
tool_param("query", "string", "search query to send", required = TRUE)
```

Index