

Package ‘rtemis.llm’

September 24, 2026

Title Large Language Models and Agentic AI

Version 0.8.7

Date 2026-09-17

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. Runs Apple's on-device 'Foundation Models' through the 'rtemis-afm' bridge <<https://github.com/rtemis-org/rtemis-afm>>.

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

Imports data.table, digest, httr2, jsonlite, jsonvalidate (>= 1.5.0),
rtemis.core (>= 0.4.6), S7

Suggests keyring, testthat (>= 3.0.0), xml2

Config/testthat/edition 3

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.0

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 2026-09-24 12:40:12 UTC

Contents

rtemis.llm-package	3
agentapply	3
anthropic_check_model	5
anthropic_list_models	6
apple_check_available	7
apple_health	8
as.list.Message	9
as_list	10
available_tools	10
config_Anthropic	11
config_Apple	12
config_Ollama	14
config_OpenAI	15
create_agent	16
create_Anthropic	18
create_Apple	19
create_custom_tool	21
create_Ollama	22
create_OpenAI	23
create_tool	25
field	26
generate	27
llmapply	29
logprobs	31
map	32
ollama_check_model	33
ollama_get_model_info	34
ollama_list_models	35
openai_check_model	35
openai_list_models	36
reasoning	37
responses	38
schema	39
token_probs	40
tools	41
tool_param	42
validate_output	43
validation_results	43

Index

45

rtemis.llm-package	rtemis.llm: <i>Agentic AI for the rtemis ecosystem</i>
--------------------	---

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]

Authors:

- 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	<i>Apply an Agent over a vector of prompts</i>
------------	--

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,
  on_error = c("na", "abort"),
  validate_output = TRUE,
  on_validation_failure = c("warn", "collect", "abort"),
  ...
)
```

Arguments

<code>x</code>	Character or list: Values to iterate over.
<code>model_or_agent</code>	Character or Agent: Either the name of a model (a string) or a pre-built Agent object from create_agent .
<code>backend</code>	Character {"ollama", "openai", "anthropic"}: Backend to use when <code>model_or_agent</code> is a string. Ignored when <code>model_or_agent</code> is an Agent object.
<code>system_prompt</code>	Character: System prompt for the on-the-fly Agent.
<code>tools</code>	Optional list of Tool objects: Tools available to the on-the-fly Agent.
<code>use_memory</code>	Logical: Whether the on-the-fly Agent should keep conversation memory.
<code>max_tool_rounds</code>	Integer [1, Inf): Maximum number of tool call rounds per query.
<code>output_schema</code>	Optional Schema: Output schema for the on-the-fly Agent.
<code>verbosity</code>	Integer [0, Inf): Verbosity level.
<code>extract_responses</code>	Logical: If TRUE, return a character vector of assistant responses. If FALSE, return the raw list of lists of Message objects.
<code>on_error</code>	Character {"na", "abort"}: What to do when a single call fails. "na" warns, keeps that element's slot as NA_character_ (or NULL when <code>extract_responses</code> = FALSE), and carries on, so a run of thousands of calls is not lost to one timeout. "abort" propagates the error and discards every result in the batch.
<code>validate_output</code>	Logical: Validate final output locally when a schema is supplied. Disabling this does not disable the schema sent to the model.
<code>on_validation_failure</code>	Character {"warn", "collect", "abort"}: Preserve invalid output and report a styled rtemis.core message (not an R warning), collect silently, or abort with an error carrying output and validation. See validate_output .
<code>...</code>	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.

Progress is reported through `rtemis.core`'s nested progress API: one status line labelled with the model name, ticking once per element, with an ETA. Set `verbosity = 0L` to silence it. When a message sink is set (see `rtemis.core::set_msg_sink()`), progress is forwarded as structured events instead of being drawn, and nests under any enclosing progress node.

Value

If `extract_responses = TRUE`, a character vector the same length as `x`. Otherwise, a list of lists of Message objects. Under `on_error = "na"` the result carries an `errors` attribute: a data.frame of index and message, one row per failed call, so failures can be retried by position rather than found by scanning for NA. With a schema, [validation_results](#) retrieves per-input statuses and diagnostics. Invalid outputs remain in place under `warn/collect`; `warn` emits one informational summary message for the batch, respecting `verbosity`.

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)
```

apple_check_available *Check the Apple Foundation Model Is Available*

Description

Reads the bridge's health with [apple_health](#) and stops with a message that says what to do when the bridge is not running, the server is not rtemis-afm, or the model is unavailable (device not eligible, Apple Intelligence turned off, model still downloading).

Usage

```
apple_check_available(base_url = APPLE_URL_DEFAULT)
```

Arguments

base_url Character: Base URL of the bridge's OpenAI-compatible wire.

Value

Named list, invisibly: The health response, if the model is available; otherwise throws an error.

Author(s)

EDG

Examples

```
# Requires a running rtemis-afm bridge
## Not run:
apple_check_available()

## End(Not run)
```

apple_health	<i>Read the rtemis-afm Bridge's Health</i>
--------------	--

Description

Reads GET /health of a running rtemis-afm bridge and returns its body: status, version, and model, which carries the served model's id, name, availability ("available" or "unavailable"), reason when unavailable ("deviceNotEligible", "appleIntelligenceNotEnabled", or "modelNotReady"), and context_window in tokens.

Usage

```
apple_health(base_url = APPLE_URL_DEFAULT)
```

Arguments

base_url Character: Base URL of the bridge's OpenAI-compatible wire. The health endpoint is read from the same server, beside /v1.

Value

Named list: The parsed health response.

Author(s)

EDG

Examples

```
# Requires a running rtemis-afm bridge
## Not run:
  apple_health()

## End(Not run)
```

as.list.Message	<i>Convert Message to List</i>
-----------------	--------------------------------

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	<i>Convert to R list</i>
---------	--------------------------

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	<i>Print built-in tools available for use by agents</i>
-----------------	---

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_Apple

Create an Apple Foundation Models Config Object

Description

Creates an AppleConfig object which can be passed to create_agent().

Usage

```
config_Apple(
  temperature = TEMPERATURE_DEFAULT,
  model_name = APPLE_MODEL_DEFAULT,
  base_url = APPLE_URL_DEFAULT,
  timeout = APPLE_TIMEOUT_DEFAULT,
```

```

    extra_headers = NULL,
    extra_body = NULL,
    validate_model = TRUE
  )

```

Arguments

temperature	Numeric [0, 2]: The temperature for the model.
model_name	Character: The model id the bridge serves; "afm" unless the bridge says otherwise.
base_url	Character: Base URL of the bridge's OpenAI-compatible wire.
timeout	Numeric (0, Inf): Request timeout in seconds.
extra_headers	Optional list: Additional HTTP headers.
extra_body	Optional list: Additional request body fields.
validate_model	Logical: Whether to check the bridge's /health endpoint now with apple_check_available , so that a bridge that is not running or a model that is not available fails here with a message saying what to do, rather than at the first request.

Details

Apple's on-device Foundation Model (the model behind Apple Intelligence) is reached through the rtemis-afm bridge, which serves it over the OpenAI Chat Completions wire on `http://127.0.0.1:1977`. Install and start the bridge with `curl -fsSL https://live.rtemis.org/afm.sh | sh` (or `brew install rtemis-org/tap/rtemis-afm`, then `rtemis-afm`). It needs an Apple silicon Mac, macOS 27 or later, and Apple Intelligence turned on. No API key is used or sent.

The model supports chat, structured output, and tool calling, with an 8,192-token context window on macOS 27.0; a prompt that does not fit is refused by the bridge with a `context_length_exceeded` error.

Value

AppleConfig object

Author(s)

EDG

Examples

```

# Requires a running rtemis-afm bridge
## Not run:
  cfg <- config_Apple(temperature = 0.2)
  agent <- create_agent(cfg, system_prompt = "You are a concise assistant.")

## End(Not run)
# Build the configuration without contacting the bridge:
cfg <- config_Apple(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. When NULL, the field is omitted from requests and Ollama uses the model default. 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,
    zero_data_retention = 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.
zero_data_retention	Optional logical: Whether to require OpenRouter to route each request only to a zero-data-retention endpoint. Supported only with an OpenRouter base URL.
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.

Details

With `zero_data_retention = TRUE`, each OpenRouter request includes `provider.zdr = true`. OpenRouter will then consider only endpoints with a ZDR policy. This option does not activate account-level ZDR at OpenAI, Anthropic, or other providers.

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
)
# Require an OpenRouter endpoint that does not retain prompts or responses:
openrouter_cfg <- config_OpenAI(
  model_name = "inclusionai/ling-3.0-flash-fin:free",
  base_url = "https://openrouter.ai/api/v1",
  api_key_env = "OPENROUTER_API_KEY",
  zero_data_retention = TRUE,
  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 , config_Anthropic , or config_Apple .
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 <code>getOption("rtemis_security_logfile", tempfile("rtemis_security_log_", fileext = ".jsonl"))</code> 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_Apple

Create an Apple Foundation Models LLM Object

Description

A stateless LLM backed by Apple's on-device Foundation Model through the rtemis-afm bridge; see [config_Apple](#) for what the bridge is and how to start it.

Usage

```
create_Apple(
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  temperature = TEMPERATURE_DEFAULT,
  output_schema = NULL,
  name = NULL,
  model_name = APPLE_MODEL_DEFAULT,
```

```

    base_url = APPLE_URL_DEFAULT,
    timeout = APPLE_TIMEOUT_DEFAULT,
    extra_headers = NULL,
    extra_body = NULL,
    validate_model = TRUE
  )

```

Arguments

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.
model_name	Character: The model id the bridge serves; "afm" unless the bridge says otherwise.
base_url	Character: Base URL of the bridge's OpenAI-compatible wire.
timeout	Numeric (0, Inf): Request timeout in seconds.
extra_headers	Optional list: Additional HTTP headers.
extra_body	Optional list: Additional request body fields.
validate_model	Logical: Whether to check the bridge's /health endpoint now with apple_check_available , so that a bridge that is not running or a model that is not available fails here with a message saying what to do, rather than at the first request.

Value

Apple LLM object

Author(s)

EDG

Examples

```

# Requires a running rtemis-afm bridge
## Not run:
  llm <- create_Apple(system_prompt = "You are a meticulous research assistant.")
  generate(llm, "What is the capital of France?")

## End(Not run)

```

create_custom_tool	<i>create_custom_tool</i>
--------------------	---------------------------

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

<code>name</code>	Character: The name of the tool, e.g. "Addition".
<code>function_name</code>	Character: The name to expose to the model, e.g. "add_numbers".
<code>description</code>	Character: The description of the tool.
<code>parameters</code>	List of ToolParameter: The parameters of the tool, each defined using tool_param .
<code>impl</code>	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. When NULL, the field is omitted from requests and Ollama uses the model default.

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,
  zero_data_retention = NULL,
  enable_thinking = NULL,
  validate_model = FALSE
)
```

Arguments

<code>model_name</code>	Character: The name of the LLM model to use.
<code>system_prompt</code>	Character: The system prompt to use.
<code>temperature</code>	Numeric [0, 2]: The temperature for the model.
<code>output_schema</code>	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.
zero_data_retention	Optional logical: Whether to require OpenRouter to route each request only to a zero-data-retention endpoint. Supported only with an OpenRouter base URL.
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.

Details

With `zero_data_retention = TRUE`, each OpenRouter request includes `provider.zdr = true`. OpenRouter will then consider only endpoints with a ZDR policy. This option does not activate account-level ZDR at OpenAI, Anthropic, or other providers.

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	<i>create_tool</i>
-------------	--------------------

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	<i>Define a schema field</i>
-------	------------------------------

Description

Define a schema field

Usage

```
field(  
  name,  
  description = name,  
  type = c("string", "number", "integer", "boolean", "array"),  
  enum = NULL,  
  items = NULL,  
  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"}: The field type. A field cannot be an object, because it carries no properties; for an array of objects, set type = "array" and items = schema(...).
enum	Optional Character: Permitted values for this field. Only for type "string", "number" or "integer". Backends that support constrained decoding (e.g. Ol-lama) make any other value impossible rather than merely detectable.
items	Character, Field or Schema: What an array field contains. Required for type = "array" and forbidden otherwise. Give a type name ("string", "number", "integer", "boolean"), a field where the elements need their own description or enum, or a schema for an array of objects.
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")
field("flag", "Whether the result is out of range", enum = c("low", "normal", "high"))

# An array of strings: one element per item, so nothing has to be delimited
# inside a single string and later split back apart.
field("questions", "Each question, quoted as written", type = "array", items = "string")

# An array whose elements carry their own description
field(
  "codes", "ICD-10 codes found",
  type = "array",
  items = field("code", "One ICD-10 code, e.g. \"E11.9\"")
)

# An array of objects
field(
  "results", "One row per lab result",
  type = "array",
  items = schema(
    "LabResult",
    field("name", "Test name"),
    field("value", "Result value", 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,
  validate_output = TRUE,
  on_validation_failure = c("warn", "collect", "abort"),
```

```
    ...  
  )
```

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 <code>options.num_predict</code> ; for OpenAI-compatible backends this maps to <code>max_tokens</code> .
stop	Optional character: Stop sequence(s). Mapped to <code>stop_sequences</code> on Anthropic and <code>options.stop</code> 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.
validate_output	Logical: Validate final output locally when a schema is supplied. Disabling this does not disable the schema sent to the model.
on_validation_failure	Character {"warn", "collect", "abort"}: Preserve invalid output and report a styled <code>remis.core</code> message (not an R warning), collect silently, or abort with an error carrying output and validation. See validate_output .
...	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), `num_ctx` (integer, mapped to `options.num_ctx`), `keep_alive` (character duration such as "10m", or seconds as a number, controlling how long the model stays loaded after the request), `logprobs` (logical), `top_logprobs` (integer)
- **OpenAI**: `seed` (integer), `logprobs` (logical), `top_logprobs` (integer [0, 20])
- **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). With a schema, [validation_results](#) retrieves the attached validation report. Invalid output is retained unless explicitly configured to abort. Agent validation checks only the final answer, before committing that answer to memory.

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,
  on_error = c("na", "abort"),
  validate_output = TRUE,
  on_validation_failure = c("warn", "collect", "abort"),
  ...
)
```

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.
on_error	Character {"na", "abort"}: What to do when a single call fails. "na" warns, keeps that element's slot as NA_character_ (or NULL when extract_responses = FALSE), and carries on, so a run of thousands of calls is not lost to one timeout. "abort" propagates the error and discards every result in the batch.
validate_output	Logical: Validate final output locally when a schema is supplied. Disabling this does not disable the schema sent to the model.
on_validation_failure	Character {"warn", "collect", "abort"}: Preserve invalid output and report a styled rtemis.core message (not an R warning), collect silently, or abort with an error carrying output and validation. See validate_output .
...	Additional per-call arguments forwarded to generate (e.g. temperature, top_p, max_tokens, stop, think, top_k, seed, and for Ollama num_ctx and keep_alive).

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.

Progress is reported through rtemis.core's nested progress API: one status line labelled with the model name, ticking once per element, with an ETA. Set verbosity = 0L to silence it. When a message sink is set (see `rtemis.core::set_msg_sink()`), progress is forwarded as structured events instead of being drawn, and nests under any enclosing progress node.

Value

If `extract_responses = TRUE`, a character vector the same length as x. Otherwise, a list of Message objects. Under `on_error = "na"` the result carries an `errors` attribute: a data.frame of index and message, one row per failed call, so failures can be retried by position rather than found by scanning for NA. With a schema, [validation_results](#) retrieves per-input statuses and diagnostics. Invalid outputs remain in place under `warn/collect`; `warn` emits one informational summary message for the batch, respecting verbosity.

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)
```

logprobs

*Extract token log probabilities from a Message or list of Messages***Description**

Returns the per-token log probabilities of the assistant's response, when the call was made with `logprobs = TRUE`. Reading a probability off the first token is better calibrated than asking a model to emit a number, so this is the basis for scoring a binary question.

Usage

```
logprobs(x, top = FALSE)
```

Arguments

<code>x</code>	Message object, list of Message objects, or list of lists of Message objects.
<code>top</code>	Logical: If TRUE, add an <code>alternatives</code> list column holding the top-k alternative tokens considered at each position, each as a <code>data.table</code> of token, logprob and prob. The alternatives are only present when the call also set <code>top_logprobs</code> .

Details

Supported on Ollama and OpenAI-compatible backends; Anthropic does not return log probabilities, so its messages yield NULL rather than an error.

Value

For a single Message, a `data.table` of position, token, logprob and prob, or NULL when the message carries no log probabilities. For a list, a list of those, one per element, so the length of the result matches the length of `x`.

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
llm <- create_ollama("gemma4:e4b", system_prompt = "Answer Yes or No only.")
msg <- generate(
  llm, "Is the sky blue?",
  think = FALSE, logprobs = TRUE, top_logprobs = 5L
)
logprobs(msg)

## End(Not run)
```

map	<i>Map</i>
-----	------------

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(), plus the two arguments the methods accept: verbosity and on_error. See Details.

Details

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

Both methods accept:

- verbosity Integer [0, Inf): Verbosity level. Progress is reported through rtemis.core’s nested progress API - one status line labelled with the model name, ticking once per element, with an ETA - and verbosity = 0L silences it. The per-call verbosity is verbosity - 1L. When a message sink is set (see rtemis.core::set_msg_sink()), progress is forwarded as structured events instead of being drawn, and nests under any enclosing progress node.
- on_error Character {"na", "abort"}: What to do when a single call fails. "na" (the default) warns, leaves NULL in that element’s slot, and carries on; the result then carries an errors attribute, a data.frame of index and message with one row per failed call. [responses](#) and [reasoning](#) map those NULL slots to NA_character_. "abort" propagates the error and discards every result in the batch.

Value

A list of Message objects (for LLM) or list of lists of Message objects (for Agent). With an output schema, [validation_results](#) retrieves an aligned report. Validation happens per item; on_validation_failure = "warn" emits one summary message at completion, respecting verbosity. Validation options are forwarded to [generate](#).

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	<i>Check Ollama Model is Available</i>
--------------------	--

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	<i>List Ollama Models</i>
--------------------	---------------------------

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	<i>Check OpenAI-compatible Model Is Available</i>
--------------------	---

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	<i>List OpenAI-compatible Models</i>
--------------------	--------------------------------------

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	<i>Extract reasoning trace(s) from a Message or list of Messages</i>
-----------	--

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. NULL elements are allowed: llmapply and agentapply leave one in the slot of every call that failed under on_error = "na".
---	---

Value

Character vector of reasoning traces, with NA_character_ in slots where no reasoning is available (including failed calls).

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	<i>Extract response(s) from a Message or list of Messages</i>
-----------	---

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. NULL elements are allowed: llmapply and agentapply leave one in the slot of every call that failed under on_error = "na".
---	---

Value

Character vector of assistant responses. Returns NA_character_ in slots where no assistant message is present (including failed calls), so that the length of the result matches the length of x. An agent history returned by [generate](#) yields only the final assistant response. Attached validation reports are preserved.

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	<i>Define output schema for LLM responses</i>
--------	---

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")
)
```

token_probs	<i>Probability of specific tokens at one position</i>
-------------	---

Description

Reads the probability of each candidate answer directly off the model's token distribution, rather than parsing a value the model emitted. Asking a binary question and taking $P(\text{"Yes"})$ at the first position is far better calibrated than asking the model for a number.

Usage

```
token_probs(x, tokens, position = 1L)
```

Arguments

x	Message object, list of Message objects, or list of lists of Message objects.
tokens	Character: Candidate tokens to report probabilities for.
position	Integer [1, Inf): Token position to read. Position 1 is the first generated token, which is the answer only when reasoning is off - a thinking model spends its opening tokens on the reasoning channel.

Details

Candidates are matched on trimmed, case-folded token text, and the probabilities of every matching alternative are summed, so "Yes" picks up " Yes" and "yes" too. A candidate that does not appear among the returned alternatives is NA, not zero: its probability is unknown, only bounded above by the smallest one returned. Raise top_logprobs if candidates you care about come back NA.

Value

For a single Message, a named numeric vector, one element per token in tokens. For a list, a numeric matrix with one row per element of x and one column per token.

Author(s)

EDG

Examples

```
# Requires running Ollama server and gemma4:e4b model
## Not run:
llm <- create_ollama("gemma4:e4b", system_prompt = "Answer Yes or No only.")
msg <- generate(
  llm, "Is the sky blue?",
  think = FALSE, logprobs = TRUE, top_logprobs = 5L
)
token_probs(msg, c("Yes", "No"))

## End(Not run)
```

tools	<i>Built-in Agent Tools</i>
-------	-----------------------------

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)
```

<code>tool_param</code>	<i>tool_param</i>
-------------------------	-------------------

Description

Define a tool parameter schema

Usage

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

Arguments

<code>name</code>	Character: The name of the parameter.
<code>type</code>	Character: The type of the parameter.
<code>description</code>	Character: The description of the parameter.
<code>required</code>	Logical: Whether the parameter is required.

Value

ToolParameter object

Author(s)

EDG

Examples

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

validate_output	<i>Validate saved structured responses</i>
-----------------	--

Description

Checks strict JSON syntax and the requested schema locally, without changing responses or logging validation failures. Does not coerce values or repair JSON. Extra properties are allowed by the current Schema; optional fields may be absent but may not be null. An object field constrains only its outer type; an array field also constrains its elements, since `items` says what it holds.

Usage

```
validate_output(x, schema)
```

Arguments

x	Character, Message, or list: JSON response text(s), a single message, an LLM batch, or a list of agent conversations. A flat message list is an LLM batch, as in responses ; wrap one agent conversation in <code>list()</code> to check only its final assistant answer. A result directly from <code>generate(agent, ...)</code> is recognized as a single conversation.
schema	Schema: Requested output schema created with schema .

Value

An S7 validation report with `@output` (original text), `@status` (valid, invalid, unavailable, or not_validated), `@schema`, and `@issues` (a data.frame with index, path, keyword, and message). Missing responses are unavailable; malformed JSON is invalid. Empty input gives an empty report.

Examples

```
sch <- schema("Count", field("n", type = "integer"))
report <- validate_output(c('{ "n":10}', '{ "n": "10" }', NA_character_), sch)
report@status
report@issues
```

validation_results	<i>Retrieve an attached validation report</i>
--------------------	---

Description

Retrieve an attached validation report

Usage

```
validation_results(x)
```

Arguments

x Message, character, or list: A generated message, extracted response(s), agent history, or batch result.

Value

An OutputValidation report, or NULL if none is attached. With a supplied schema and disabled validation, the attached report records not_validated.

Examples

```
validation_results("unvalidated text")
```

Index

agentapply, [3](#), [37](#), [38](#)
anthropic_check_model, [5](#)
anthropic_list_models, [6](#)
apple_check_available, [7](#), [13](#), [20](#)
apple_health, [7](#), [8](#)
as.list.Message, [9](#)
as_list, [10](#)
available_tools, [10](#)

config_Anthropic, [11](#), [17](#)
config_Apple, [12](#), [17](#), [19](#)
config_Ollama, [14](#), [17](#)
config_OpenAI, [15](#), [17](#)
create_agent, [4](#), [16](#), [21](#)
create_agent(), [41](#)
create_Anthropic, [18](#), [29](#)
create_Apple, [19](#)
create_custom_tool, [17](#), [21](#)
create_Ollama, [22](#), [29](#)
create_OpenAI, [23](#), [29](#)
create_tool, [21](#), [25](#)

field, [17](#), [26](#), [26](#), [39](#)

generate, [4](#), [17](#), [27](#), [30](#), [33](#), [38](#)

llmapply, [5](#), [29](#), [37](#), [38](#)
logprobs, [31](#)

map, [32](#)

ollama_check_model, [33](#)
ollama_get_model_info, [34](#)
ollama_list_models, [35](#)
openai_check_model, [35](#)
openai_list_models, [36](#)

reasoning, [32](#), [37](#)
responses, [32](#), [38](#), [43](#)
rtemis.llm(rtemis.llm-package), [3](#)
rtemis.llm-package, [3](#)

schema, [17](#), [18](#), [20](#), [22](#), [23](#), [26](#), [30](#), [39](#), [43](#)

token_probs, [40](#)
tool_arxiv(tools), [41](#)
tool_datetime(tools), [41](#)
tool_duckduckgo_ia(tools), [41](#)
tool_param, [21](#), [25](#), [42](#)
tool_semanticscholar(tools), [41](#)
tool_wikipedia(tools), [41](#)
tools, [41](#)

validate_output, [4](#), [28](#), [30](#), [43](#)
validation_results, [5](#), [28](#), [30](#), [33](#), [43](#)