Package 'rtemis.llm'

November 3, 2026

Title Large Language Models and Agentic AI

Version 0.8.8

Date 2026-11-02

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, image input, 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>. Fills closed schemas and answers typed questions with decision models served by 'Ollama' and 'OpenRouter' <https://openrouter.ai>.

License BSD_3_clause + file LICENSE

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, httr2, jsonlite, jsonvalidate (>= 1.5.0), openssl, rtemis.core (>= 0.4.6), S7

Suggests mirai (>= 2.7.0), pkgload, processx, withr, keyring, testthat (>= 3.0.0), xml2

Config/testthat/edition 3

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.1

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 2026-11-03 00:20:02 UTC

Contents

rtemis.llm-package . . . 3
agentapply . . . 4
anthropic_check_model . . . 6
anthropic_list_models . . . 7
apple_check_available . . . 8
apple_health . . . 9
as.list.Message . . . 9
as_list . . . 10
as_questions . . . 11
available_tools . . . 12
chat . . . 13
choice . . . 14
config_Anthropic . . . 15
config_Apple . . . 17
config_Ollama . . . 18
config_OllamaDecision . . . 20
config_OpenAI . . . 21
config_OpenRouterDecision . . . 22
create_agent . . . 23
create_Anthropic . . . 25
create_Apple . . . 27
create_custom_tool . . . 28
create_DecisionModel . . . 29
create_Ollama . . . 31
create_OpenAI . . . 32
create_tool . . . 34
decide . . . 35
dmapply . . . 37
field . . . 39
generate . . . 40
is_decidable . . . 42
llmapply . . . 43
logprobs . . . 45
map . . . 46
noul . . . 48
ollama_check_model . . . 49
ollama_get_model_info . . . 50
ollama_list_decision_models . . . 50
ollama_list_models . . . 51
openai_check_model . . . 52
openai_list_models . . . 53
probabilities . . . 54
reasoning . . . 56
responses . . . 57
schema . . . 58
token_probs . . . 58

tools 59
tool_param 61
validate_output 61
validation_results 62

Index **63**

`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]

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 *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,
  on_error = c("na", "abort"),
  validate_output = TRUE,
  on_validation_failure = c("warn", "collect", "abort"),
  image_path = NULL,
  concurrency = 1L,
  ...
)
```

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.

`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.

`image_path`
: Optional character or list: Local image files to send with the prompts. A character vector sends one image with each element of `x`; a list of character vectors sends several (a `NULL` entry sends none). Length 1 is recycled across `x`, and a single prompt in `x` is recycled across `image_path`, so one question can be asked of many images. Every file is checked before the first request. See generate.

`concurrency`
: Positive integer: Maximum number of independent input tasks running at once. Defaults to 1 (sequential). Values greater than 1 require the optional 'mirai' package. See `map()` for agent isolation, worker requirements, and provider limits.

`...`
: 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. Within-call tool history is retained with either memory setting. With `concurrency > 1L`, every input creates a fresh agent; the supplied agent's accumulated history is neither copied nor modified. See `map()`.

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 *Read the rtemis-afm Bridge's Health*

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 *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)
```

as_questions *Convert a Schema to Decision Model Questions*

Description

Returns the questions a decision model is asked when it fills `x` with `generate()`. Each is worded from the field's name and description, and named after the field. Edit them, for example by giving a choice's options their meanings, and ask them with `decide()` or `dmapply(questions = ...)`.

Usage

```
as_questions(x)
```

Arguments

`x`	Schema: A closed schema (see `is_decidable()`).

Details

The conversion:

- A field with an `enum` becomes a `choice()` named after the field. Its options are the `enum` values, each meaning only its own label.
- A boolean field becomes a `noul()` named after the field.
- An array whose `items` is a field with an `enum` becomes one `noul()` per value, named `<field>[<value>]`.
- A field that is not required adds a `noul()` named `<field>?`, asking whether the passage gives it at all.

A choice takes at most 26 options. `generate()` fills an `enum` with more values by asking about groups of values and then choosing among the group winners. That has no single question to return, so it is an error here.

Value

Named list of `Choice` and `Noul` objects, in field order.

Author(s)

EDG

Examples

```
sentiment <- schema(
  "Sentiment",
 field("sentiment", "Sentiment of the sentence", enum = c("positive", "negative", "neutral"))
)
qs <- as_questions(sentiment)
qs
```

```
# Give the options their meanings, keeping the wording
qs[["sentiment"]] <- choice(
  qs[["sentiment"]]@instructions,
  c(
    positive = "The writer is pleased or approving overall",
    negative = "The writer is displeased or critical overall",
    neutral = "The writer states facts, or balances praise and criticism evenly"
  )
)
```

`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()
```

chat *Chat with an LLM or Agent in the console*

Description

chat runs a multi-turn conversation in the R console. Each line you type is sent with generate to an Agent that keeps the conversation in memory, and the reply is printed before the next prompt, until you type /exit.

Usage

```
chat(
  x,
  backend = c("ollama", "openai", "anthropic"),
  system_prompt = SYSTEM_PROMPT_DEFAULT,
  tools = NULL,
  max_tool_rounds = 3L,
  verbosity = 1L,
  ...
)
```

Arguments

x	Character, LLM, or Agent: A model name, an LLM, or an Agent.
backend	Character {"ollama", "openai", "anthropic"}: Backend to use when x is a model name.
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.
max_tool_rounds	Integer [1, Inf): Maximum number of tool call rounds per message, for the on-the-fly Agent.
verbosity	Integer [0, Inf): Verbosity level. At 1, the banner, progress and tool calls are shown; at 0, only the replies.
...	Additional per-call arguments forwarded to generate on every turn, such as temperature or think.

Details

Pass a model name (an Agent is built on the fly using backend, system_prompt, tools, max_tool_rounds), an LLM from create_Ollama, create_OpenAI, create_Anthropic, or create_Apple (wrapped in an Agent with memory, using the LLM's configuration, system prompt, and output schema), or an Agent from create_agent (used as given; its memory must be on). Chatting with an existing Agent continues its conversation, and the Agent is updated in place.

Lines starting with / are commands and are not sent to the model:

- `/image <path>`: attach a local image to the next message; repeat to attach several. See `image_path` in generate.
- `/clear`: forget the conversation, keeping the system prompt.
- `/help`: list the commands.
- `/exit` or `/quit`: end the chat. Ctrl-C (Esc in RStudio) at the prompt also ends it.

Interrupting while a reply is on its way, or a failed request, discards that turn: the message and any partial exchange are removed from memory and the chat continues.

Value

The `Agent`, invisibly, with the conversation in its memory.

Author(s)

EDG

Examples

```
# Requires an interactive session, a running Ollama server and the gemma4:e4b model
## Not run:
  agent <- chat("gemma4:e4b")
  # After /exit, the conversation is in the agent's memory
  agent

## End(Not run)
```

`choice` *Define a Choice Question*

Description

A question a decision model answers by choosing one of 2 to 26 named options, with a probability for each.

Usage

```
choice(instructions, options)
```

Arguments

`instructions`	Character: The question.
`options`	Character: The options. Name the vector to say what each option means (`c(low = "Can wait a week")`); an unnamed option means its own name.

Value

`Choice` object, for `decide()`.

Author(s)

EDG

Examples

```
choice(
  "Which team should handle this ticket?",
  c(billing = "Payments, invoices, refunds", engineering = "Bugs and crashes")
)
```

---

config_Anthropic *Create a AnthropicConfig Object*

Description

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

Usage

```
config_Anthropic(
  model_name,
  temperature = 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,
  think = NULL,
  effort = NULL,
  fallbacks = NULL,
  validate_model = FALSE
)
```

Arguments

`model_name`	Character: The name of the Anthropic model to use (for example `"claude-opus-5-5"`).
`temperature`	Optional numeric [0, 1]: The temperature for the model. `NULL` omits it from the request, and the model uses its default. Claude Opus 4.7 and later reject it; Claude Sonnet 5, Claude Sonnet 5.5 and Claude Haiku 5.5 accept only `1`, their default.
`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, thinking included. 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, sent as `thinking: {type: "enabled", budg` on models that accept a budget (Claude Opus 4.6, Claude Sonnet 4.6, and Claude 4.5 models and earlier). Must be less than `max_tokens`.

`think` Optional logical: Whether the model thinks. `NULL` runs the model's default. `TRUE` sends `thinking_budget_tokens` when set, and adaptive thinking otherwise. `FALSE` sends the model's off form: `{type: "disabled"}`, `{type: "between_tools"}` on Claude Sonnet 5.5, or no `thinking` field. Claude Opus 5.5 and Claude Fable 5 models always think, and `FALSE` aborts. Can be overridden per call.

`effort` Optional character {"low", "medium", "high", "xhigh", "max"}: How much the model thinks and writes, sent as `output_config.effort`. `NULL` runs the model's default (`"medium"` on Claude Opus 5.5 and Claude Haiku 5.5, `"high"` on other models with effort). Can be overridden per call.

`fallbacks` Optional character: Server-side fallbacks for refused requests: `"default"` (Anthropic chooses the fallback model by refusal category), or 1 to 3 model names. `NULL` sends no fallback, so every response comes from `model_name`. Claude Sonnet 5.5 accepts only `"default"`. The matching `anthropic-beta` header is added.

`validate_model` Logical: Whether to check the model with the Models API and read its capabilities.

Details

Every request follows the request rules of the model named in `model_name`: which sampling parameters it accepts, how thinking is turned on and off, its effort levels, whether structured output uses `output_config.format` or a forced tool call, whether it accepts a trailing assistant message, and whether it has server-side fallbacks. A value the model rejects aborts here or before the request, with a message naming the model and the argument. The rules come from a table built into the package. Model names absent from the table, such as those of Anthropic-compatible servers, send every configured field as given and use a forced tool call for structured output. With `validate_model = TRUE`, the model's capabilities read from the Models API (`GET /models/{id}`) replace the table's structured-output, thinking and effort entries.

A response with stop_reason "refusal" raises an error of class rtemis_llm_refusal, whose category and explanation fields carry the response's stop_details. A response that stops at max_tokens raises a warning and returns the truncated text.

Value

AnthropicConfig object

Author(s)

EDG

Examples

```
cfg <- config_Anthropic(
   model_name = "claude-opus-5-5",
   effort = "low",
   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,
  schema_in_prompt = TRUE
)
```

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.
schema_in_prompt	Logical: Whether to append the output schema to the system prompt of each request that has one. Ollama uses the schema, sent as format, only to constrain decoding, so the model sees field descriptions only through the prompt. The system prompt is followed by a blank line, Respond with JSON that conforms to this JSON Schema:, and the schema as JSON. Set to FALSE when the system prompt already states the schema and the field instructions.

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_OllamaDecision` *Configure a Decision Model on Ollama*

Description

A decision model answers typed questions with a probability for every option and writes no text. Ollama serves the ones it lists with the `decision` capability through `/v1/systemone`.

Usage

```
config_OllamaDecision(
  model_name,
  base_url = OLLAMA_URL_DEFAULT,
  timeout = OLLAMA_DECISION_TIMEOUT_DEFAULT,
  validate_model = TRUE
)
```

Arguments

`model_name`	Character: The decision model's name on the Ollama server, for example `"clef-flash:9b"`.
`base_url`	Character: Base URL of the Ollama server.
`timeout`	Numeric (0, Inf): Request timeout in seconds.
`validate_model`	Logical: If `TRUE`, check that the server has the model and that it is a decision model.

Value

`OllamaDecisionConfig` object, to pass to `create_DecisionModel()`.

Author(s)

EDG

Examples

```
# Requires a running Ollama server with a decision model
## Not run:
  config_OllamaDecision("clef-flash:9b")

## 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
)
```

---

config_OpenRouterDecision

Configure a Decision Model on OpenRouter

Description

OpenRouter serves decision models (for example `"typesafe/jev-1.13"`) through the same `/v1/systemone` contract as Ollama. The request, including the prompt and any images, leaves the computer.

Usage

```
config_OpenRouterDecision(
  model_name,
  api_key = NULL,
  api_key_env = OPENROUTER_API_KEY_ENV_DEFAULT,
  keychain_service = NULL,
  base_url = OPENROUTER_URL_DEFAULT,
  timeout = OPENROUTER_DECISION_TIMEOUT_DEFAULT,
  extra_headers = NULL
)
```

Arguments

model_name	Character: The decision model's name on OpenRouter.
api_key	Optional Character: API key.
api_key_env	Character: Environment variable holding the API key.
keychain_service	Optional Character: macOS Keychain service holding the API key.
base_url	Character: Base URL; requests go to <base_url>/v1/systemone.
timeout	Numeric (0, Inf): Request timeout in seconds.
extra_headers	Optional named list: Extra HTTP headers.

Details

The API key is resolved at request time: api_key if given, then the variable named by api_key_env or the Keychain item named by keychain_service, then OPENROUTER_API_KEY.

Value

OpenRouterDecisionConfig object, to pass to create_DecisionModel().

Author(s)

EDG

Examples

```
config_OpenRouterDecision("typesafe/jev-1.13")
```

---

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 retain conversation history across calls. A call retains its own messages and tool results with either setting.
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 = NULL,
  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,
  think = NULL,
  effort = NULL,
  fallbacks = NULL,
  validate_model = FALSE
)
```

Arguments

`model_name`	Character: The name of the Anthropic model to use (for example `"claude-opus-5-5"`).
`system_prompt`	Character: The system prompt to use.
`temperature`	Optional numeric [0, 1]: The temperature for the model. `NULL` omits it from the request, and the model uses its default. Claude Opus 4.7 and later reject it; Claude Sonnet 5, Claude Sonnet 5.5 and Claude Haiku 5.5 accept only `1`, their default.
`output_schema`	Optional Schema: Output schema created using schema. On models with structured outputs, the schema is sent as `output_config.format` and the answer is the text block of the response; on other models, structured output is a forced call of a synthetic tool 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. A character vector is comma-joined.
max_tokens	Integer [1, Inf): Maximum number of tokens the model may generate, thinking included. 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, sent as thinking: {type: "enabled", budg on models that accept a budget (Claude Opus 4.6, Claude Sonnet 4.6, and Claude 4.5 models and earlier). Must be less than max_tokens.
think	Optional logical: Whether the model thinks. NULL runs the model's default. TRUE sends thinking_budget_tokens when set, and adaptive thinking otherwise. FALSE sends the model's off form: {type: "disabled"}, {type: "between_tools"} on Claude Sonnet 5.5, or no thinking field. Claude Opus 5.5 and Claude Fable 5 models always think, and FALSE aborts. Can be overridden per call.
effort	Optional character {"low", "medium", "high", "xhigh", "max"}: How much the model thinks and writes, sent as output_config.effort. NULL runs the model's default ("medium" on Claude Opus 5.5 and Claude Haiku 5.5, "high" on other models with effort). Can be overridden per call.
fallbacks	Optional character: Server-side fallbacks for refused requests: "default" (Anthropic chooses the fallback model by refusal category), or 1 to 3 model names. NULL sends no fallback, so every response comes from model_name. Claude Sonnet 5.5 accepts only "default". The matching anthropic-beta header is added.
validate_model	Logical: Whether to check the model with the Models API and read its capabilities.

Details

See config_Anthropic for how each request follows the configured model's rules.

Value

Anthropic LLM object

Author(s)

EDG

Examples

```
llm <- create_Anthropic(
   model_name = "claude-opus-5-5",
   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.

Details

The on-device model declines some requests. The bridge returns each refusal as a message with a `refusal` field, which raises an error of class `rtemis_llm_refusal`; under `on_error = "na"`, llmapply and map leave `NA` in that element's slot and record it in the `errors` attribute.

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` *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_DecisionModel *Create a Decision Model*

Description

A decision model answers typed questions about a passage with a probability for every option, and writes no text. There are two kinds of question: a `choice()` among 2 to 26 named options, and a `noul()`, which asks how true a statement is.

Usage

```
create_DecisionModel(config, context = NULL, output_schema = NULL, name = NULL)
```

Arguments

`config`	`DecisionConfig`: From `config_OllamaDecision()` or `config_OpenRouterDecision()`.
`context`	Optional Character: Text placed before every prompt, in the state the questions are judged against. A decision model has no system prompt; put the task's standing context here.
`output_schema`	Optional Schema: The closed schema `generate()` fills.
`name`	Optional Character: Name.

Details

Questions are the decision model's own interface. Ask them with `decide()`, or over many passages with `dmapply(questions = ...)`. You write each question's wording and, for a choice, what each option means. The model judges the passage against those meanings, so well-defined options are usually the most accurate way to use a decision model.

A schema is an adapter for code written for LLMs. `generate()` and `dmapply()` can also fill a closed `schema()`, returning a validated JSON document in the shape an LLM's structured output has. The schema is turned into questions first. `as_questions()` returns them, so you can read them, edit them and pass them to `decide()`.

- A field with an `enum` becomes a `choice()`, asked as "Which value of the field <name> (<description>) does the passage support?". Each option means only its own label.
- A boolean field becomes a `noul()`.
- An array whose `items` is a field with an `enum` becomes one `noul()` per value. The values at or above 0.5 are kept.
- A field that is not required adds a `noul()` asking whether the passage gives it at all.

Use a schema when the result must be a document: to drop a decision model into a pipeline built for LLMs, or to validate the output. Use questions when the answer you want is the judgment itself, and to give options their meanings. Use `is_decidable()` to check whether a schema is closed.

Value

`DecisionModel` object.

Author(s)

EDG

Examples

```
# Requires a running Ollama server with a decision model
## Not run:
  triage <- schema(
    "Triage",
    field("team", "Team to route to", enum = c("billing", "engineering", "sales")),
    field("urgent", "Whether it needs an answer today", type = "boolean")
  )
 dm <- create_DecisionModel(config_OllamaDecision("clef-flash:9b"), output_schema = triage)
  generate(dm, "The app crashes whenever I open the billing page.")

## End(Not run)
```

`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,
  schema_in_prompt = TRUE
)
```

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. Ollama constrains the response to the schema's structure, types, and `enum` values. With `schema_in_prompt = TRUE`, the schema, including field descriptions, is also appended to the system prompt of each request.
`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.
`schema_in_prompt`	Logical: Whether to append the output schema to the system prompt of each request that has one. Ollama uses the schema, sent as `format`, only to constrain decoding, so the model sees field descriptions only through the prompt. The system prompt is followed by a blank line, `Respond with JSON that conforms to this JSON Schema:`, and the schema as JSON. Set to `FALSE` when the system prompt already states the schema and the field instructions.

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

`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.
`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.

A response whose message carries a non-empty `refusal`, or an error response with code `"refusal"`, raises an error of class `rtemis_llm_refusal`, whose `refusal`, `finish_reason`, and `status_code` fields carry the server's refusal text, the choice's `finish_reason`, and the HTTP status.

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
    )
  )
)
```

`decide` *Ask a Decision Model Typed Questions*

Description

Sends `questions` about `state` to a decision model. Each `choice()` is answered with the option chosen, a probability per option, and the provider's confidence (how concentrated the probabilities are); each `noul()` with the probability that it is true. More than 64 questions are sent as several calls.

Usage

```
decide(x, state, questions, image_path = NULL, verbosity = 1L)
```

Arguments

`x`	`DecisionModel`: From `create_DecisionModel()`.
`state`	Character: The passage the questions are about. The model's `context`, if any, is placed before it.
`questions`	Named list of `choice()` and `noul()` questions. The names identify the answers. `as_questions()` converts a schema into such a list.
`image_path`	Optional Character: Paths to local images (PNG, JPEG or WebP) judged with the state.
`verbosity`	Integer: Verbosity level.

Value

`Decision` object. Use `probabilities()` for a table of every option's probability.

Probability and confidence

A choice is answered with a probability for each option, given the passage, the question's wording and the options' meanings. The probabilities sum to 1 over the options asked. The answer is the option with the highest probability, and `p` is that option's probability.

Its `confidence` is computed by the provider from those probabilities: one minus their normalized Shannon entropy,

$$c = 1 + \frac{\sum_i p_i \log p_i}{\log n}$$

over the n options asked, with $0 \log 0 = 0$. It is 1 when one option holds all the probability and 0 when the probability is spread evenly. It describes the shape of the distribution and adds no information beyond the probabilities.

A noul is answered with `p`, the probability that the statement is true; `1 - p` is the probability that it is false. A noul has no `confidence`: the distance of `p` from 0.5 shows how decided the answer is.

Reading them. These terms read differently in statistics:

- `p` is the probability of an answer, not a p-value. No hypothesis is tested, and a `p` near 1 supports the answer.
- `p` is the model's probability, as the provider returns it. It is a frequency only if the model is calibrated for the task: among answers given `p = 0.8`, 80% are right. Calibration is measured on labeled passages, for example as the proportion correct in bins of `p`.
- `p` compares only the options asked. Adding, removing or redefining an option changes every probability, and an answer outside the options has no probability.
- `confidence` is not the probability that the choice is right, and not a confidence level as in a confidence interval. It measures how concentrated the probability is over all options, so it depends on the number of options and on how the remaining probability is spread. With two options, `p = 0.8` gives a confidence of 0.28, and a confidence of 0.5 needs `p = 0.89`. With four options, `p = 0.6` gives 0.51 when the other 0.4 is on one option and 0.20 when it is spread over three. A tie at 0.5 between two of six options gives 0.61. Compare the top two probabilities in `probabilities()` to find close calls.

Unsure. An answer is flagged `unsure` when a choice's confidence is below 0.5, or a noul's `p` is between 0.35 and 0.65. The thresholds are fixed constants; set thresholds for a task from labeled passages.

Author(s)

EDG

Examples

```
# Requires a running Ollama server with a decision model
## Not run:
  dm <- create_DecisionModel(config_OllamaDecision("clef-flash:9b"))
  decide(
    dm,
    "The app crashes whenever I open the billing page. Please fix it today.",
    list(
     team = choice("Which team should handle this?", c("billing", "engineering", "sales")),
       urgent = noul("Does the customer need an answer today?")
    )
  )

## End(Not run)
```

dmapply *Apply a Decision Model over a vector of prompts*

Description

dmapply is the lapply-style entry point for a decision model: it fills a closed schema from each element of x, or, with questions, asks the same typed questions about each. Pass either a model name (in which case a DecisionModel is built on the fly using backend, context and output_schema) or a pre-built DecisionModel from create_DecisionModel().

Usage

```
dmapply(
  x,
  model_or_dm,
  backend = c("ollama", "openrouter"),
  context = NULL,
  output_schema = NULL,
  questions = NULL,
  verbosity = 1L,
  extract_responses = TRUE,
  on_error = c("na", "abort"),
  validate_output = TRUE,
  on_validation_failure = c("warn", "collect", "abort"),
  image_path = NULL,
  concurrency = 1L
)
```

Arguments

x	Character or list: Values to iterate over. Each element is the passage for one call.
model_or_dm	Character or DecisionModel: The name of a decision model, or a pre-built DecisionModel.
backend	Character {"ollama", "openrouter"}: Provider to use when model_or_dm is a string. Ignored when model_or_dm is a DecisionModel.
context	Optional Character: Text placed before every passage, when building the DecisionModel from a name.
output_schema	Optional Schema: The closed schema to fill, when building the DecisionModel from a name. Supplying it with a pre-built DecisionModel is a conflict and will error.
questions	Optional named list of choice() and noul() questions. When given, each element of x is answered with decide() instead of filling a schema.
verbosity	Integer [0, Inf): Verbosity level. The per-call verbosity is verbosity - 1L.

`extract_responses`
: Logical: If `TRUE`, return the filled documents as a character vector of JSON (or, with `questions`, the `probabilities()` table). If `FALSE`, return the list of `DecisionMessage` (or `Decision`) objects.

`on_error`
: Character {"na", "abort"}: What to do when a single call fails, as in `llmapply()`.

`validate_output`
: Logical: Validate each filled document against the schema.

`on_validation_failure`
: Character {"warn", "collect", "abort"}: As in `generate()`.

`image_path`
: Optional character or list: Local images judged with the passages, recycled as in `llmapply()`.

`concurrency`
: Positive integer: Maximum number of independent input tasks running at once. Defaults to 1 (sequential). Values greater than 1 require the optional 'mirai' package. See `map()` for agent isolation, worker requirements, and provider limits.

Details

Progress, errors and validation work as in `llmapply()`. The result carries an `elapsed` attribute: the wall time of each call, in seconds, as `llmapply()` and `agentapply()` record it, so the speed of a decision model and an LLM can be compared item by item.

Value

With `extract_responses = TRUE`, a character vector the same length as `x` (or a `data.table` with `questions`). Otherwise a list of `DecisionMessage` (or `Decision`) objects. The result carries an `elapsed` attribute (seconds per element) and, under `on_error = "na"`, an `errors` attribute as in `llmapply()`. With a schema, `validation_results()` retrieves per-input statuses.

Author(s)

EDG

Examples

```
# Requires a running Ollama server with a decision model
## Not run:
  tickets <- c(
    "The app crashes whenever I open the billing page.",
    "Can I get a quote for 50 seats?"
  )
  triage <- schema(
    "Triage",
    field("team", "Team to route to", enum = c("billing", "engineering", "sales")),
    field("urgent", "Whether it needs an answer today", type = "boolean")
  )
  res <- dmapply(tickets, "clef-flash:9b", output_schema = triage)
  attr(res, "elapsed")

## End(Not run)
```

`field` *Define a schema field*

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. Apple Foundation Models uses the description to guide generation. Ollama uses an output schema only to constrain decoding, so the model sees the description through the prompt: an Ollama LLM appends the schema to the system prompt (see `schema_in_prompt` in create_Ollama).
`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. Ollama) 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,
  image_path = 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, Agent or DecisionModel. A DecisionModel fills its output schema and takes no sampling options; see create_DecisionModel().
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.
image_path	Optional character: Paths to one or more local image files (PNG, JPEG, GIF, or WebP) to send with the prompt. Each file is checked and read before the request is sent. The model must accept image input; a model that does not is reported as an error by its server.
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 rtemis.core 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)
```

---

is_decidable *Check Whether a Decision Model Can Fill a Schema*

Description

A decision model fills only closed fields: a field with an enum (a choice), a boolean (a noul), or an array whose items is a field with an enum (one noul per value). A field that is not required adds a question about whether the passage gives it at all. Free text, numbers and integers without an enum, and arrays of anything else are open, and keep the schema an LLM's.

Usage

```
is_decidable(x)
```

Arguments

x	Schema: Created with schema().

Value

Logical scalar, with two attributes: open, a data.frame of the open fields (path, reason) in field order; and questions, the number of questions a fill asks before any tournament final (NA when the schema is open).

Author(s)

EDG

Examples

```
is_decidable(schema(
  "Triage",
  field("team", enum = c("billing", "engineering")),
  field("summary", "One-line summary")
))
```

`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"),
  image_path = NULL,
  concurrency = 1L,
  ...
)
```

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. On the Ollama backend the schema constrains the structure, types, and `enum` values of each response and is appended to the system prompt, so the model sees field descriptions (see `schema_in_prompt` in create_Ollama).

`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.

`image_path` Optional character or list: Local image files to send with the prompts. A character vector sends one image with each element of `x`; a list of character vectors sends several (a `NULL` entry sends none). Length 1 is recycled across `x`, and a single prompt in `x` is recycled across `image_path`, so one question can be asked of many images. Every file is checked before the first request. See generate.

`concurrency` Positive integer: Maximum number of independent input tasks running at once. Defaults to 1 (sequential). Values greater than 1 require the optional 'mirai' package. See `map()` for agent isolation, worker requirements, and provider limits.

`...` 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
  )
  # One question over many images, with a structured answer per image
  llmapply(
    "Describe this figure.",
    "gemma4:e4b",
    output_schema = schema(
      "Figure",
      field("chart type", "Type of chart"),
      field("series", "Number of data series", type = "integer")
    ),
    image_path = c("fig1.png", "fig2.png")
  )

## 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

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

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 *Map*

Description

Map

Usage

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

Arguments

x	A character vector or list to map over.
f	An LLM, Agent or DecisionModel object.
...	Additional arguments passed to generate(), plus the method arguments verbosity, on_error, and concurrency. See Details.

Details

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

All 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.
- concurrency Positive integer: Maximum number of independent input tasks active at once. Defaults to 1. Values greater than 1 require 'mirai' and use a private pool of local R workers. Results retain input order even when requests finish out of order.

Concurrent agent calls are independent. Each input creates a new agent with its own conversation state, initialized from the supplied agent's configuration and system prompt. Messages, provider-returned reasoning, and tool results remain available throughout that input's run. Accumulated history from the supplied agent is not copied; its state remains unchanged. This also applies to successive inputs handled by one worker. Use extract_responses = FALSE in agentapply() to retain each run's message history. With concurrency = 1L, an agent can retain conversation history across inputs as configured.

Workers reuse the normal generation path. Tool functions and captured data must be serializable; the caller's global environment and live connections are not available in workers. Use package-qualified function calls and explicitly captured data in custom tools. Tools must coordinate shared external resources themselves. Agent security records are appended to the configured log by the main R process. Nested concurrent mapping is rejected.

Client concurrency bounds input tasks, including all their model requests and tool rounds. Provider limits apply separately: Ollama's OLLAMA_NUM_PARALLEL controls per-model server concurrency; the Apple bridge uses --concurrency; cloud APIs enforce account and model quotas. Increasing client concurrency can add queueing without improving throughput. Concurrent requests retry HTTP 429 and 503 up to three total attempts, honoring Retry-After or using backoff with jitter. A shared cooldown pauses further model requests within the batch. Retries have a 60-second budget after the first transient response; a longer server delay ends that input's retries. Model requests are retried individually, preserving completed tool steps. Transport failures follow on_error without

automatic replay. Abort stops dispatch and cancels remaining workers; completed external effects remain.

Results carry `elapsed` (seconds per input, including its retry waits, excluding worker startup and time awaiting dispatch) and `batch_elapsed` (wall time for mapping, including worker startup and shutdown). Overlapping input durations must not be summed to estimate batch wall time. Progress is emitted by the main R process as inputs finish.

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)
```

`noul` *Define a Noul Question*

Description

A question a decision model answers with the probability that a statement is true.

Usage

```
noul(instructions)
```

Arguments

`instructions`	Character: The statement or yes/no question.

Value

Noul object, for decide().

Author(s)

EDG

Examples

```
noul("Does the customer ask for a refund?")
```

---

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_decision_models *List Decision Models on Ollama*

Description

List Decision Models on Ollama

Usage

```
ollama_list_decision_models(base_url = OLLAMA_URL_DEFAULT)
```

Arguments

base_url	Character: Base URL of the Ollama server.

Value

Character vector: Names of the models whose capabilities include `decision`.

Author(s)

EDG

Examples

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

## 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)
```

`probabilities` *Extract a Decision Model's Probabilities*

Description

Tabulates every option's probability from a `decide()` result, a schema filled by `generate()`, or a list of either from `dmapply()`.

Usage

```
probabilities(x)
```

Arguments

`x`	`Decision`, `DecisionMessage`, or a list of them (`NULL` elements, left by failed calls, are skipped).

Value

`data.table` with one row per option: `index` (the element's position, for a list), `question` (a `Decision`'s question name) or `path` (a filled field), `option`, `p`, `chosen` and `unsure`. A noul has two rows, `"true"` and `"false"`. An array field has one row per value, `p` being the probability that it belongs.

Filled schemas

A field of a schema filled by `generate()` reports the probability of the value written:

- An `enum` field is a choice: `p` is the chosen value's probability, and the `options` sum to 1.
- A boolean field is a noul: the value is `true` when the noul's `p` is at least 0.5, and the field's `p` is the probability of the value written, so it is at least 0.5.
- An array field is one noul per value. Each value's `p` is the probability that it belongs, from its own question, so the values' probabilities need not sum to 1. Values with `p` at least 0.5 are kept.
- An `enum` with more than 26 values is decided by a tournament. Its `p` is from the final, among the group winners only.

- An `enum` with one value is decided by elimination, without a question: `p` is 1 and the field is never unsure.
- A field that is not required is left out when its presence noul is below 0.5, and is listed in the message's `omitted` property.

Probability and confidence

A choice is answered with a probability for each option, given the passage, the question's wording and the options' meanings. The probabilities sum to 1 over the options asked. The answer is the option with the highest probability, and `p` is that option's probability.

Its `confidence` is computed by the provider from those probabilities: one minus their normalized Shannon entropy,

$$c = 1 + \frac{\sum_i p_i \log p_i}{\log n}$$

over the n options asked, with $0 \log 0 = 0$. It is 1 when one option holds all the probability and 0 when the probability is spread evenly. It describes the shape of the distribution and adds no information beyond the probabilities.

A noul is answered with `p`, the probability that the statement is true; `1 - p` is the probability that it is false. A noul has no `confidence`: the distance of `p` from 0.5 shows how decided the answer is.

Reading them. These terms read differently in statistics:

- `p` is the probability of an answer, not a p-value. No hypothesis is tested, and a `p` near 1 supports the answer.
- `p` is the model's probability, as the provider returns it. It is a frequency only if the model is calibrated for the task: among answers given `p = 0.8`, 80% are right. Calibration is measured on labeled passages, for example as the proportion correct in bins of `p`.
- `p` compares only the options asked. Adding, removing or redefining an option changes every probability, and an answer outside the options has no probability.
- `confidence` is not the probability that the choice is right, and not a confidence level as in a confidence interval. It measures how concentrated the probability is over all options, so it depends on the number of options and on how the remaining probability is spread. With two options, `p = 0.8` gives a confidence of 0.28, and a confidence of 0.5 needs `p = 0.89`. With four options, `p = 0.6` gives 0.51 when the other 0.4 is on one option and 0.20 when it is spread over three. A tie at 0.5 between two of six options gives 0.61. Compare the top two probabilities in `probabilities()` to find close calls.

Unsure. An answer is flagged `unsure` when a choice's confidence is below 0.5, or a noul's `p` is between 0.35 and 0.65. The thresholds are fixed constants; set thresholds for a task from labeled passages.

Author(s)

EDG

Examples

```
# Requires a running Ollama server with a decision model
## Not run:
  dm <- create_DecisionModel(config_OllamaDecision("clef-flash:9b"))
 d <- decide(dm, "Please refund my last invoice.", list(refund = noul("Is this a refund?")))
  probabilities(d)

## 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. `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` *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. `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 *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")
)
```

token_probs *Probability of specific tokens at one position*

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("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 *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)
```

validate_output *Validate saved structured responses*

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 `list()` to check only its final assistant answer. A result directly from `generate(agent, ...)` 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 *Retrieve an attached validation report*

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, 4, *56, 57*
agentapply(), *38*, *47*
anthropic_check_model, 6
anthropic_list_models, 7
apple_check_available, 8, *18*, *27*
apple_health, *8*, 9
as.list.Message, 9
as_list, 10
as_questions, 11
as_questions(), *30*, *35*
available_tools, 12

chat, 13
choice, 14
choice(), *11*, *29*, *35*, *37*
config_Anthropic, 15, *24*, *26*
config_Apple, 17, *24*, *27*
config_Ollama, 18, *24*
config_OllamaDecision, 20
config_OllamaDecision(), *29*
config_OpenAI, 21, *24*
config_OpenRouterDecision, 22
config_OpenRouterDecision(), *29*
create_agent, *4*, *13*, 23, *28*
create_agent(), *59*
create_Anthropic, *13*, 25, *43*
create_Apple, *13*, 27
create_custom_tool, *24*, 28
create_DecisionModel, 29
create_DecisionModel(), *20*, *23*, *35*, *37*, *41*
create_Ollama, *13*, 31, *39*, *43, 44*
create_OpenAI, *13*, 32, *43*
create_tool, *28*, 34

decide, 35
decide(), *11*, *14*, *30*, *37*, *49*, *54*
dmapply, 37
dmapply(), *30*, *54*

field, *24*, *39*, 39, *58*

generate, *5*, *13, 14*, *24*, 40, *44*, *48*, *57*
generate(), *11*, *29, 30*, *38*, *54*

is_decidable, 42
is_decidable(), *11*, *30*

llmapply, *5*, *28*, 43, *56, 57*
llmapply(), *38*
logprobs, 45

map, *28*, 46
map(), *5*, *38*, *44*

noul, 48
noul(), *11*, *29*, *35*, *37*

ollama_check_model, 49
ollama_get_model_info, 50
ollama_list_decision_models, 50
ollama_list_models, 51
openai_check_model, 52
openai_list_models, 53

probabilities, 54
probabilities(), *35, 36*, *38*, *55*

reasoning, *47*, 56
responses, *47*, 57, *62*
rtemis.llm *(*rtemis.llm-package*)*, 3
rtemis.llm-package, 3

schema, *24, 25*, *27*, *31*, *33*, *39*, *44*, 58, *62*
schema(), *30*, *42*

token_probs, 58
tool_arxiv *(*tools*)*, 59
tool_datetime *(*tools*)*, 59
tool_duckduckgo_ia *(*tools*)*, 59
tool_param, *28*, *34*, 61
tool_semanticscholar *(*tools*)*, 59
tool_wikipedia *(*tools*)*, 59

tools, 59

validate_output, *5*, *41*, *44*, 61
validation_results, *5*, *42*, *45*, *48*, 62
validation_results(), *38*