Package 'mini007'

December 4, 2025

Type Package

Title Lightweight Framework for Orchestrating Multi-Agent Large
Language Models

Version 0.2.2

Description Provides tools for creating agents with persistent state using
R6 classes <https://cran.r-project.org/package=R6> and the 'ellmer'
package <https://cran.r-project.org/package=ellmer>. Tracks prompts,
messages, and agent metadata for reproducible, multi-turn large language
model sessions.

License MIT + file LICENSE

Encoding UTF-8

Imports checkmate (>= 2.3.1), cli (>= 3.6.5), DiagrammeR (>= 1.0.11),
ellmer (>= 0.4.0), glue (>= 1.8.0), R6 (>= 2.6.1), uuid (>=
1.2.0)

RoxygenNote 7.3.2

NeedsCompilation no

Author Mohamed El Fodil Ihaddaden [aut, cre]

Maintainer Mohamed El Fodil Ihaddaden <ihaddaden.fodeil@gmail.com>

Repository CRAN

Date/Publication 2025-12-04 12:20:02 UTC

Contents

Agent   *Agent: A General-Purpose LLM Agent*

Description

The 'Agent' class defines a modular LLM-based agent capable of responding to prompts using a defined role/instruction. It wraps an OpenAI-compatible chat model via the ['ellmer'](https://github.com/llrs/ellmer) package.

Each agent maintains its own message history and unique identity.

Public fields

name The agent's name.

instruction The agent's role/system prompt.

llm_object The underlying 'ellmer::chat_openai' object.

agent_id A UUID uniquely identifying the agent.

model_provider The name of the entity providing the model (eg. OpenAI)

model_name The name of the model to be used (eg. gpt-4.1-mini)

broadcast_history A list of all past broadcast interactions.

budget A budget in $ that the agent should not exceed.

budget_policy A list controlling budget behavior: on_exceed and warn_at.

budget_warned Internal flag indicating whether warn_at notice was emitted.

cost The current cost of the agent

Active bindings

messages Public active binding for the conversation history. Assignment is validated automatically.

Methods

Public methods:

- [Agent$new()](#)
- [Agent$invoke()](#)
- [Agent$generate_execute_r_code()](#)
- [Agent$set_budget()](#)
- [Agent$set_budget_policy()](#)
- [Agent$keep_last_n_messages()](#)
- [Agent$add_message()](#)
- [Agent$clear_and_summarise_messages()](#)
- [Agent$update_instruction()](#)
- [Agent$get_usage_stats()](#)

- `Agent$reset_conversation_history()`
- `Agent$export_messages_history()`
- `Agent$load_messages_history()`
- `Agent$validate_response()`
- `Agent$clone_agent()`
- `Agent$clone()`

Method new(): Initializes a new Agent with a specific role/instruction.

Usage:

```
Agent$new(name, instruction, llm_object, budget = NA)
```

Arguments:

name A short identifier for the agent (e.g. '"translator"').

instruction The system prompt that defines the agent's role.

llm_object The LLM object generate by ellmer (eg. output of ellmer::chat_openai)

budget Numerical value denoting the amount to set for the budget in US$ to a specific agent, if the budget is reached, an error will be thrown.

Examples:

```
# An API KEY is required in order to invoke the Agent
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

polar_bear_researcher <- Agent$new(
  name = "POLAR BEAR RESEARCHER",
  instruction = paste0(
  "You are an expert in polar bears, ",
 "you task is to collect information about polar bears. Answer in 1 sentence max."
  ),
  llm_object = openai_4_1_mini
)
```

Method invoke(): Sends a user prompt to the agent and returns the assistant's response.

Usage:

```
Agent$invoke(prompt)
```

Arguments:

prompt A character string prompt for the agent to respond to.

Returns: The LLM-generated response as a character string.

Examples:

```
\dontrun{
# An API KEY is required in order to invoke the Agent
openai_4_1_mini <- ellmer::chat(
```

```
      name = "openai/gpt-4.1-mini",
      api_key = Sys.getenv("OPENAI_API_KEY"),
      echo = "none"
 )
 agent <- Agent$new(
  name = "translator",
  instruction = "You are an Algerian citizen",
  llm_object = openai_4_1_mini
 )
 agent$invoke("Continue this sentence: 1 2 3 viva")
 }
```

Method generate_execute_r_code(): Generate R code from natural language descriptions and optionally validate/execute it

Usage:
```
 Agent$generate_execute_r_code(
   code_description,
   validate = FALSE,
   execute = FALSE,
   interactive = TRUE,
   env = globalenv()
 )
```

Arguments:

code_description Character string describing the R code to generate

validate Logical indicating whether to validate the generated code syntax

execute Logical indicating whether to execute the generated code (use with caution)

interactive Logical; if TRUE, ask for user confirmation before executing generated code

env Environment in which to execute the code if execute = TRUE. Default to globalenv

Returns: A list containing the generated code and validation/execution results

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
r_assistant <- Agent$new(
  name = "R Code Assistant",
  instruction = paste("You are an expert R programmer",
  llm_object = openai_4_1_mini
)
# Generate code for data manipulation
result <- r_assistant$generate_execute_r_code(
  code_description = "Calculate the summary of the mtcars dataframe",
  validate = TRUE,
  execute = TRUE,
```

```
    interactive = TRUE
)
print(result)
}
```

Method `set_budget()`: Set a budget to a specific agent, if the budget is reached, an error will be thrown

Usage:
```
Agent$set_budget(amount_in_usd)
```

Arguments:

`amount_in_usd` Numerical value denoting the amount to set for the budget,

Examples:
```
\dontrun{
# An API KEY is required in order to invoke the Agent
openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
)
agent <- Agent$new(
 name = "translator",
 instruction = "You are an Algerian citizen",
 llm_object = openai_4_1_mini
)
agent$set_budget(amount_in_usd = 10.5) # this is equivalent to 10.5$
}
```

Method `set_budget_policy()`: Configure how the agent behaves as it approaches or exceeds its budget. Use 'warn_at' (0-1) to emit a one-time warning when spending reaches the specified fraction of the budget. When the budget is exceeded, 'on_exceed' controls behavior: abort, warn and proceed, or ask interactively.

Usage:
```
Agent$set_budget_policy(on_exceed = "abort", warn_at = 0.8)
```

Arguments:

`on_exceed` One of "abort", "warn", or "ask".

`warn_at` Numeric in (0,1); fraction of budget to warn at. Default 0.8.

Examples:
```
\dontrun{
agent$set_budget(5)
agent$set_budget_policy(on_exceed = "ask", warn_at = 0.9)
}
```

Method `keep_last_n_messages()`: Keep only the most recent 'n' messages, discarding older ones while keeping the system prompt.

Usage:

```
Agent$keep_last_n_messages(n = 2)
```

Arguments:

n Number of most recent messages to keep.

Examples:

```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "capital finder",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$invoke("What is the capital of Algeria")
agent$invoke("What is the capital of Germany")
agent$invoke("What is the capital of Italy")
agent$keep_last_n_messages(n = 2)
}
```

Method `add_message()`: Add a pre-formatted message to the conversation history

Usage:

```
Agent$add_message(role, content)
```

Arguments:

role The role of the message ("user", "assistant", or "system")

content The content of the message

Examples:

```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "AI assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$add_message("user", "Hello, how are you?")
agent$add_message("assistant", "I'm doing well, thank you!")
}
```

Method `clear_and_summarise_messages()`: Summarises the agent's conversation history into a concise form and appends it to the system prompt. Unlike 'update_instruction()', this method does not override the existing instruction but augments it with a summary for future context.

After creating the summary, the method clears the conversation history and retains only the updated system prompt. This ensures that subsequent interactions start fresh but with the summary preserved as context.

Usage:
```
Agent$clear_and_summarise_messages()
```
Examples:
```
\dontrun{
  # Requires an OpenAI-compatible LLM from `ellmer`
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )

  agent <- Agent$new(
    name = "summariser",
    instruction = "You are a summarising assistant",
    llm_object = openai_4_1_mini
  )

  agent$invoke("The quick brown fox jumps over the lazy dog.")
  agent$invoke("This is another example sentence.")

  # Summarises and resets history
  agent$summarise_messages()

  # Now only the system prompt (with summary) remains
  agent$messages
}
```

Method update_instruction(): Update the system prompt/instruction

Usage:
```
Agent$update_instruction(new_instruction)
```
Arguments:

new_instruction New instruction to use. Not that the new instruction will override the old one

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "assistant",
```

```
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$update_instruction("You are a concise assistant.")
}
```

Method `get_usage_stats()`: Get the current token count and estimated cost of the conversation

Usage:
```
Agent$get_usage_stats()
```

Returns: A list with token counts and cost information

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$set_budget(1)
agent$invoke("What is the capital of Algeria?")
stats <- agent$get_usage_stats()
stats
}
```

Method `reset_conversation_history()`: Reset the agent's conversation history while keeping the system instruction

Usage:
```
Agent$reset_conversation_history()
```

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "AI assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$invoke("Hello, how are you?")
agent$invoke("Tell me about machine learning")
agent$reset_conversation_history()  # Clears all messages except system prompt
}
```

Method `export_messages_history()`: Saves the agent's current conversation history as a JSON file on disk.

Usage:
```
Agent$export_messages_history(
  file_path = paste0(getwd(), "/", paste0(self$name, "_messages.json"))
)
```

Arguments:

`file_path` Character string specifying the file path where the JSON file should be saved. Defaults to a file named '"<agent_name>_messages.json"' in the current working directory.

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "capital_finder",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$invoke("What is the capital of Algeria")
agent$invoke("What is the capital of Italy")
agent$export_messages_history()
}
```

Method `load_messages_history()`: Load an agent's conversation history as a JSON file from disk.

Usage:
```
Agent$load_messages_history(
  file_path = paste0(getwd(), "/", paste0(self$name, "_messages.json"))
)
```

Arguments:

`file_path` Character string specifying the file path where the JSON file is stored. Defaults to a file named '"<agent_name>_messages.json"' in the current working directory.

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "capital_finder",
```

```
    instruction = "You are an assistant.",
    llm_object = openai_4_1_mini
  )
  # use the export_messages_history to save the interaction first
  agent$load_messages_history("path/to/messages.json")
  agent$messages
  agent$llm_object
  }
```

Method `validate_response()`:   Validates an agent's response against custom criteria using
LLM-based validation. This method uses the agent's LLM to evaluate whether a response meets
specified validation criteria (e.g., accuracy, completeness, tone, format).

Usage:
```
Agent$validate_response(
  prompt,
  response,
  validation_criteria,
  validation_score = 0.8
)
```

Arguments:

`prompt` The prompt used to generate the response.

`response` The response text to validate.

`validation_criteria` The criteria for validation. (e.g., "The response should be accurate and
    complete", "The response must be under 100 words").

`validation_score` A numeric from 0 to 1 denoting the score to consider for the evaluation.
    During the evaluation, the agent will provide a score from 0 to 1, a provided score greater
    or equal to the 'validation_score' will result in a 'valid' response. Defaults to 0.8.

Returns: list object

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "fact_checker",
  instruction = "You are a factual assistant.",
  llm_object = openai_4_1_mini
)
prompt <- "What is the capital of Algeria?"
response <- agent$invoke(prompt)
validation <- agent$validate_response(
  response = response,
  prompt = prompt,
```

```
    validation_criteria = "The response must be accurate and mention Algiers",
    validation_score = 0.8
  )
  print(validation)
  }
```

Method `clone_agent()`: Create a copy of the agent with the same instruction and configuration but a new unique ID. Useful for creating multiple instances of the same agent type.

Usage:
```
Agent$clone_agent(new_name = NULL)
```

Arguments:

new_name Optional character string to assign a new name to the cloned agent. If NULL, the cloned agent retains the original name.

Returns: A new Agent instance with the same configuration but a unique ID.

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "translator",
  instruction = "You are a translator.",
  llm_object = openai_4_1_mini
)
# Clone with same name
agent_copy <- agent$clone_agent()
}
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
```
Agent$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

[load_messages_history()] for reloading a saved message history.

[export_messages_history()] for exporting the messages object to json.

Examples

```
## -----------------------------------------------
## Method `Agent$new`
## -----------------------------------------------
```

```
  # An API KEY is required in order to invoke the Agent
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )

  polar_bear_researcher <- Agent$new(
    name = "POLAR BEAR RESEARCHER",
    instruction = paste0(
    "You are an expert in polar bears, ",
    "you task is to collect information about polar bears. Answer in 1 sentence max."
    ),
    llm_object = openai_4_1_mini
  )


## ----------------------------------------------
## Method `Agent$invoke`
## ----------------------------------------------

## Not run:
# An API KEY is required in order to invoke the Agent
openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
)
agent <- Agent$new(
 name = "translator",
 instruction = "You are an Algerian citizen",
 llm_object = openai_4_1_mini
)
agent$invoke("Continue this sentence: 1 2 3 viva")

## End(Not run)

## ----------------------------------------------
## Method `Agent$generate_execute_r_code`
## ----------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
r_assistant <- Agent$new(
  name = "R Code Assistant",
  instruction = paste("You are an expert R programmer",
  llm_object = openai_4_1_mini
)
```

```
# Generate code for data manipulation
result <- r_assistant$generate_execute_r_code(
  code_description = "Calculate the summary of the mtcars dataframe",
  validate = TRUE,
  execute = TRUE,
  interactive = TRUE
)
print(result)

## End(Not run)

## ------------------------------------------------
## Method `Agent$set_budget`
## ------------------------------------------------

## Not run:
# An API KEY is required in order to invoke the Agent
openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
)
agent <- Agent$new(
 name = "translator",
 instruction = "You are an Algerian citizen",
 llm_object = openai_4_1_mini
)
agent$set_budget(amount_in_usd = 10.5) # this is equivalent to 10.5$

## End(Not run)

## ------------------------------------------------
## Method `Agent$set_budget_policy`
## ------------------------------------------------

## Not run:
agent$set_budget(5)
agent$set_budget_policy(on_exceed = "ask", warn_at = 0.9)

## End(Not run)

## ------------------------------------------------
## Method `Agent$keep_last_n_messages`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "capital finder",
```

```
    instruction = "You are an assistant.",
    llm_object = openai_4_1_mini
)
agent$invoke("What is the capital of Algeria")
agent$invoke("What is the capital of Germany")
agent$invoke("What is the capital of Italy")
agent$keep_last_n_messages(n = 2)

## End(Not run)

## ------------------------------------------------
## Method `Agent$add_message`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "AI assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$add_message("user", "Hello, how are you?")
agent$add_message("assistant", "I'm doing well, thank you!")

## End(Not run)

## ------------------------------------------------
## Method `Agent$clear_and_summarise_messages`
## ------------------------------------------------

## Not run:
  # Requires an OpenAI-compatible LLM from `ellmer`
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )

  agent <- Agent$new(
    name = "summariser",
    instruction = "You are a summarising assistant",
    llm_object = openai_4_1_mini
  )

  agent$invoke("The quick brown fox jumps over the lazy dog.")
  agent$invoke("This is another example sentence.")

  # Summarises and resets history
  agent$summarise_messages()
```

```
  # Now only the system prompt (with summary) remains
  agent$messages

## End(Not run)


## ------------------------------------------------
## Method `Agent$update_instruction`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$update_instruction("You are a concise assistant.")

## End(Not run)

## ------------------------------------------------
## Method `Agent$get_usage_stats`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$set_budget(1)
agent$invoke("What is the capital of Algeria?")
stats <- agent$get_usage_stats()
stats

## End(Not run)

## ------------------------------------------------
## Method `Agent$reset_conversation_history`
## ------------------------------------------------

## Not run:
```

```
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "AI assistant",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$invoke("Hello, how are you?")
agent$invoke("Tell me about machine learning")
agent$reset_conversation_history()  # Clears all messages except system prompt

## End(Not run)


## ---------------------------------------------
## Method `Agent$export_messages_history`
## ---------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "capital_finder",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
agent$invoke("What is the capital of Algeria")
agent$invoke("What is the capital of Italy")
agent$export_messages_history()

## End(Not run)


## ---------------------------------------------
## Method `Agent$load_messages_history`
## ---------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "capital_finder",
  instruction = "You are an assistant.",
  llm_object = openai_4_1_mini
)
```

```
# use the export_messages_history to save the interaction first
agent$load_messages_history("path/to/messages.json")
agent$messages
agent$llm_object

## End(Not run)


## ------------------------------------------------
## Method `Agent$validate_response`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "fact_checker",
  instruction = "You are a factual assistant.",
  llm_object = openai_4_1_mini
)
prompt <- "What is the capital of Algeria?"
response <- agent$invoke(prompt)
validation <- agent$validate_response(
  response = response,
  prompt = prompt,
  validation_criteria = "The response must be accurate and mention Algiers",
  validation_score = 0.8
)
print(validation)

## End(Not run)

## ------------------------------------------------
## Method `Agent$clone_agent`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
agent <- Agent$new(
  name = "translator",
  instruction = "You are a translator.",
  llm_object = openai_4_1_mini
)
# Clone with same name
agent_copy <- agent$clone_agent()
```

```
## End(Not run)
```

LeadAgent   *LeadAgent: A Multi-Agent Orchestration Coordinator*

Description

'LeadAgent' extends 'Agent' to coordinate a group of specialized agents. It decomposes complex prompts into subtasks using LLMs and assigns each subtask to the most suitable registered agent. The lead agent handles response chaining, where each agent can consider prior results.

Details

This class builds intelligent multi-agent workflows by delegating sub-tasks using 'delegate_prompt()', executing them with 'invoke()', and storing the results in the 'agents_interaction' list.

Super class

[mini007::Agent](#) -> LeadAgent

Public fields

agents A named list of registered sub-agents (by UUID).

agents_interaction A list of delegated task history with agent IDs, prompts, and responses.

plan A list containing the most recently generated task plan.

hitl_steps The steps where the workflow should be stopped in order to allow for a human interaction

prompt_for_plan The prompt used to generate the plan.

agents_for_plan The agents used for the plan

Methods

Public methods:

- [LeadAgent$new()](#)
- [LeadAgent$clear_agents()](#)
- [LeadAgent$remove_agents()](#)
- [LeadAgent$register_agents()](#)
- [LeadAgent$visualize_plan()](#)
- [LeadAgent$invoke()](#)
- [LeadAgent$generate_plan()](#)
- [LeadAgent$broadcast()](#)
- [LeadAgent$set_hitl()](#)
- [LeadAgent$judge_and_choose_best_response()](#)
- [LeadAgent$clone()](#)

Method new(): Initializes the LeadAgent with a built-in task-decomposition prompt.

Usage:

```
LeadAgent$new(name, llm_object)
```

Arguments:

name A short name for the coordinator (e.g. '"lead"').

llm_object The LLM object generate by ellmer (eg. output of ellmer::chat_openai)

Examples:

```
  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )

 lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
 )
```

Method clear_agents(): Clear out the registered Agents

Usage:

```
LeadAgent$clear_agents()
```

Examples:

```
  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
   "You are a research assistant. ",
  "Your job is to answer factual questions with detailed and accurate information. ",
   "Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
 )

  summarizer <- Agent$new(
    name = "summarizer",
    instruction = paste0(
    "You are an agent designed to summarise ",
    "a given text into 3 distinct bullet points."
```

```
  ),
  llm_object = openai_4_1_mini
)

translator <- Agent$new(
  name = "translator",
  instruction = "Your role is to translate a text from English to German",
  llm_object = openai_4_1_mini
)
lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
)

lead_agent$register_agents(c(researcher, summarizer, translator))

lead_agent$agents

lead_agent$clear_agents()

lead_agent$agents
```

Method remove_agents(): Remove registered agents by IDs

Usage:

```
LeadAgent$remove_agents(agent_ids)
```

Arguments:

agent_ids The Agent ID to remove from the registered Agents

Examples:

```
  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
   "You are a research assistant. ",
  "Your job is to answer factual questions with detailed and accurate information. ",
   "Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
)

 summarizer <- Agent$new(
   name = "summarizer",
```

```
  instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
  llm_object = openai_4_1_mini
)

translator <- Agent$new(
  name = "translator",
  instruction = "Your role is to translate a text from English to German",
  llm_object = openai_4_1_mini
)


lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
)

lead_agent$register_agents(c(researcher, summarizer, translator))

lead_agent$agents

# deleting the translator agent

id_translator_agent <- translator$agent_id

lead_agent$remove_agents(id_translator_agent)

lead_agent$agents
```

Method `register_agents()`: Register one or more agents for delegation.

Usage:

```
LeadAgent$register_agents(agents)
```

Arguments:

agents A vector of 'Agent' objects to register.

Examples:

```
  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
   "You are a research assistant. ",
  "Your job is to answer factual questions with detailed and accurate information. ",
   "Do not answer with more than 2 lines"
```

```
    ),
    llm_object = openai_4_1_mini
  )

  summarizer <- Agent$new(
    name = "summarizer",
   instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
    llm_object = openai_4_1_mini
  )

  translator <- Agent$new(
    name = "translator",
    instruction = "Your role is to translate a text from English to German",
    llm_object = openai_4_1_mini
  )

  lead_agent <- LeadAgent$new(
   name = "Leader",
   llm_object = openai_4_1_mini
  )

  lead_agent$register_agents(c(researcher, summarizer, translator))

  lead_agent$agents
```

Method `visualize_plan()`: Visualizes the orchestration plan Each agent node is shown in sequence (left → right), with tooltips showing the actual prompt delegated to that agent.

Usage:
```
 LeadAgent$visualize_plan()
```

Method `invoke()`: Executes the full prompt pipeline: decomposition → delegation → invocation.

Usage:
```
 LeadAgent$invoke(prompt, force_regenerate_plan = FALSE)
```

Arguments:

`prompt` The complex user instruction to process.

`force_regenerate_plan` If TRUE, regenerate a plan even if one exists, defaults to FALSE.

Returns: The final response (from the last agent in the sequence).

Examples:
```
\dontrun{
 # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
```

```
      researcher <- Agent$new(
        name = "researcher",
        instruction = paste0(
        "You are a research assistant. ",
        "Your job is to answer factual questions with detailed ",
        "and accurate information. Do not answer with more than 2 lines"
        ),
        llm_object = openai_4_1_mini
      )

      summarizer <- Agent$new(
        name = "summarizer",
       instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
        llm_object = openai_4_1_mini
      )

      translator <- Agent$new(
        name = "translator",
        instruction = "Your role is to translate a text from English to German",
        llm_object = openai_4_1_mini
      )

      lead_agent <- LeadAgent$new(
       name = "Leader",
       llm_object = openai_4_1_mini
      )

      lead_agent$register_agents(c(researcher, summarizer, translator))

      lead_agent$invoke(
      paste0(
       "Describe the economic situation in Algeria in 3 sentences. ",
       "Answer in German"
       )
      )
     }
```

Method `generate_plan()`: Generates a task execution plan without executing the subtasks. It returns a structured list containing the subtask, the selected agent, and metadata.

Usage:

```
LeadAgent$generate_plan(prompt)
```

Arguments:

`prompt` A complex instruction to be broken into subtasks.

Returns: A list of lists containing agent_id, agent_name, model_name, model_provider, and the assigned prompt.

Examples:

```
\dontrun{
```

```
# An API KEY is required in order to invoke the agents
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
researcher <- Agent$new(
  name = "researcher",
  instruction = paste0(
  "You are a research assistant. Your job is to answer factual questions ",
 "with detailed and accurate information. Do not answer with more than 2 lines"
  ),
  llm_object = openai_4_1_mini
)

summarizer <- Agent$new(
  name = "summarizer",
 instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
  llm_object = openai_4_1_mini
)

translator <- Agent$new(
  name = "translator",
  instruction = "Your role is to translate a text from English to German",
  llm_object = openai_4_1_mini
)

lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
)

lead_agent$register_agents(c(researcher, summarizer, translator))

lead_agent$generate_plan(
paste0(
 "Describe the economic situation in Algeria in 3 sentences. ",
 "Answer in German"
 )
)
}
```

Method broadcast()**:** Broadcasts a prompt to all registered agents and collects their responses.
This does not affect the main agent orchestration logic or history.

Usage:
```
LeadAgent$broadcast(prompt)
```
Arguments:

prompt A user prompt to send to all agents.

Returns: A list of responses from all agents.

Examples:

```
\dontrun{
 # An API KEY is required in order to invoke the agents
openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
openai_4_1 <- ellmer::chat(
  name = "openai/gpt-4.1",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

openai_4_1_agent <- Agent$new(
  name = "openai_4_1_agent",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1
)

openai_4_1_nano <- ellmer::chat(
  name = "openai/gpt-4.1-nano",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

openai_4_1_nano_agent <- Agent$new(
  name = "openai_4_1_nano_agent",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1_nano
  )

 lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
 )

lead_agent$register_agents(c(openai_4_1_agent, openai_4_1_nano_agent))
lead_agent$broadcast(
  prompt = paste0(
    "If I were Algerian, which song would I like to sing ",
    "when running under the rain? how about a flower?"
  )
  )
}
```

Method `set_hitl()`: Set Human In The Loop (HITL) interaction at determined steps within the workflow

Usage:

```
LeadAgent$set_hitl(steps)
```

Arguments:

steps At which steps the Human In The Loop is required?

Returns: A list of responses from all agents.

Examples:

```
\dontrun{
 # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
     "You are a research assistant. ",
    "Your job is to answer factual questions with detailed and accurate information. ",
     "Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
 )

 summarizer <- Agent$new(
   name = "summarizer",
   instruction = paste0(
   "You are agent designed to summarise a give text ",
   "into 3 distinct bullet points."
   ),
   llm_object = openai_4_1_mini
 )

 translator <- Agent$new(
   name = "translator",
   instruction = "Your role is to translate a text from English to German",
   llm_object = openai_4_1_mini
 )

 lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
 )

 lead_agent$register_agents(c(researcher, summarizer, translator))

 # setting a human in the loop in step 2
 lead_agent$set_hitl(1)
```

```
# The execution will stop at step 2 and a human will be able
# to either accept the answer, modify it or stop the execution of
# the workflow

lead_agent$invoke(
paste0(
 "Describe the economic situation in Algeria in 3 sentences. ",
 "Answer in German"
 )
 )
}
```

Method `judge_and_choose_best_response()`: The Lead Agent send a prompt to its registered agents and choose the best response from the agents' responses

Usage:
`LeadAgent$judge_and_choose_best_response(prompt)`

Arguments:

prompt The prompt to send to the registered agents

Returns: A list of responses from all agents, including the chosen response

Examples:
```
\dontrun{
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
openai_4_1 <- ellmer::chat(
  name = "openai/gpt-4.1",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

stylist <- Agent$new(
  name = "stylist",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1
)

openai_4_1_nano <- ellmer::chat(
  name = "openai/gpt-4.1-nano",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

stylist2 <- Agent$new(
  name = "stylist2",
```

```
    instruction = "You are an AI assistant. Answer in 1 sentence max.",
    llm_object = openai_4_1_nano
  )

  lead_agent <- LeadAgent$new(
    name = "Leader",
    llm_object = openai_4_1_mini
  )

  lead_agent$register_agents(c(stylist, stylist2))

  lead_agent$judge_and_choose_best_response("what's the best way to war a kalvin klein shirt?")

  }
```

Method clone(): The objects of this class are cloneable with this method.

Usage:
```
LeadAgent$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LeadAgent$new`
## ------------------------------------------------


  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )

 lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
 )



## ------------------------------------------------
## Method `LeadAgent$clear_agents`
## ------------------------------------------------

  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
```

```
    echo = "none"
 )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
   "You are a research assistant. ",
   "Your job is to answer factual questions with detailed and accurate information. ",
   "Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
 )

 summarizer <- Agent$new(
   name = "summarizer",
   instruction = paste0(
   "You are an agent designed to summarise ",
   "a given text into 3 distinct bullet points."
   ),
   llm_object = openai_4_1_mini
 )

 translator <- Agent$new(
   name = "translator",
   instruction = "Your role is to translate a text from English to German",
   llm_object = openai_4_1_mini
 )
 lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
 )

 lead_agent$register_agents(c(researcher, summarizer, translator))

 lead_agent$agents

 lead_agent$clear_agents()

 lead_agent$agents


## -----------------------------------------------
## Method `LeadAgent$remove_agents`
## -----------------------------------------------

 # An API KEY is required in order to invoke the agents
 openai_4_1_mini <- ellmer::chat(
   name = "openai/gpt-4.1-mini",
   api_key = Sys.getenv("OPENAI_API_KEY"),
   echo = "none"
 )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
```

```
  "You are a research assistant. ",
  "Your job is to answer factual questions with detailed and accurate information. ",
  "Do not answer with more than 2 lines"
  ),
  llm_object = openai_4_1_mini
)

summarizer <- Agent$new(
  name = "summarizer",
 instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
  llm_object = openai_4_1_mini
)

translator <- Agent$new(
  name = "translator",
  instruction = "Your role is to translate a text from English to German",
  llm_object = openai_4_1_mini
)


lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
)

lead_agent$register_agents(c(researcher, summarizer, translator))

lead_agent$agents

# deleting the translator agent

id_translator_agent <- translator$agent_id

lead_agent$remove_agents(id_translator_agent)

lead_agent$agents


## ------------------------------------------------
## Method `LeadAgent$register_agents`
## ------------------------------------------------

  # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
  name = "researcher",
  instruction = paste0(
  "You are a research assistant. ",
  "Your job is to answer factual questions with detailed and accurate information. ",
```

```
    "Do not answer with more than 2 lines"
    ),
    llm_object = openai_4_1_mini
 )

 summarizer <- Agent$new(
   name = "summarizer",
  instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
   llm_object = openai_4_1_mini
 )

 translator <- Agent$new(
   name = "translator",
   instruction = "Your role is to translate a text from English to German",
   llm_object = openai_4_1_mini
 )

 lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
 )

 lead_agent$register_agents(c(researcher, summarizer, translator))

 lead_agent$agents

## ------------------------------------------------
## Method `LeadAgent$invoke`
## ------------------------------------------------

## Not run:
 # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
   "You are a research assistant. ",
   "Your job is to answer factual questions with detailed ",
   "and accurate information. Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
 )

 summarizer <- Agent$new(
   name = "summarizer",
  instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
   llm_object = openai_4_1_mini
 )
```

```
translator <- Agent$new(
  name = "translator",
  instruction = "Your role is to translate a text from English to German",
  llm_object = openai_4_1_mini
)

lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
)

lead_agent$register_agents(c(researcher, summarizer, translator))

lead_agent$invoke(
paste0(
 "Describe the economic situation in Algeria in 3 sentences. ",
 "Answer in German"
 )
 )

## End(Not run)

## ------------------------------------------------
## Method `LeadAgent$generate_plan`
## ------------------------------------------------

## Not run:
 # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
   "You are a research assistant. Your job is to answer factual questions ",
   "with detailed and accurate information. Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
 )

 summarizer <- Agent$new(
   name = "summarizer",
  instruction = "You are agent designed to summarise a given text into 3 distinct bullet points.",
   llm_object = openai_4_1_mini
 )

 translator <- Agent$new(
   name = "translator",
   instruction = "Your role is to translate a text from English to German",
   llm_object = openai_4_1_mini
 )
```

```
lead_agent <- LeadAgent$new(
 name = "Leader",
 llm_object = openai_4_1_mini
 )

lead_agent$register_agents(c(researcher, summarizer, translator))

lead_agent$generate_plan(
 paste0(
  "Describe the economic situation in Algeria in 3 sentences. ",
  "Answer in German"
  )
 )

## End(Not run)

## ---------------------------------------------
## Method `LeadAgent$broadcast`
## ---------------------------------------------

## Not run:
 # An API KEY is required in order to invoke the agents
openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
openai_4_1 <- ellmer::chat(
  name = "openai/gpt-4.1",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

openai_4_1_agent <- Agent$new(
  name = "openai_4_1_agent",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1
)

openai_4_1_nano <- ellmer::chat(
  name = "openai/gpt-4.1-nano",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

openai_4_1_nano_agent <- Agent$new(
  name = "openai_4_1_nano_agent",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1_nano
  )

 lead_agent <- LeadAgent$new(
```

```
  name = "Leader",
  llm_object = openai_4_1_mini
 )

lead_agent$register_agents(c(openai_4_1_agent, openai_4_1_nano_agent))
lead_agent$broadcast(
  prompt = paste0(
    "If I were Algerian, which song would I like to sing ",
    "when running under the rain? how about a flower?"
  )
  )

## End(Not run)

## ------------------------------------------------
## Method `LeadAgent$set_hitl`
## ------------------------------------------------

## Not run:
 # An API KEY is required in order to invoke the agents
  openai_4_1_mini <- ellmer::chat(
    name = "openai/gpt-4.1-mini",
    api_key = Sys.getenv("OPENAI_API_KEY"),
    echo = "none"
  )
 researcher <- Agent$new(
   name = "researcher",
   instruction = paste0(
    "You are a research assistant. ",
    "Your job is to answer factual questions with detailed and accurate information. ",
    "Do not answer with more than 2 lines"
   ),
   llm_object = openai_4_1_mini
 )

 summarizer <- Agent$new(
   name = "summarizer",
   instruction = paste0(
   "You are agent designed to summarise a give text ",
   "into 3 distinct bullet points."
   ),
   llm_object = openai_4_1_mini
 )

 translator <- Agent$new(
   name = "translator",
   instruction = "Your role is to translate a text from English to German",
   llm_object = openai_4_1_mini
 )

 lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
```

```
 )

 lead_agent$register_agents(c(researcher, summarizer, translator))

 # setting a human in the loop in step 2
 lead_agent$set_hitl(1)

 # The execution will stop at step 2 and a human will be able
 # to either accept the answer, modify it or stop the execution of
 # the workflow

 lead_agent$invoke(
 paste0(
  "Describe the economic situation in Algeria in 3 sentences. ",
  "Answer in German"
  )
 )

## End(Not run)

## ------------------------------------------------
## Method `LeadAgent$judge_and_choose_best_response`
## ------------------------------------------------

## Not run:
openai_4_1_mini <- ellmer::chat(
  name = "openai/gpt-4.1-mini",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)
openai_4_1 <- ellmer::chat(
  name = "openai/gpt-4.1",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

stylist <- Agent$new(
  name = "stylist",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1
)

openai_4_1_nano <- ellmer::chat(
  name = "openai/gpt-4.1-nano",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  echo = "none"
)

stylist2 <- Agent$new(
  name = "stylist2",
  instruction = "You are an AI assistant. Answer in 1 sentence max.",
  llm_object = openai_4_1_nano
)
```

```
lead_agent <- LeadAgent$new(
  name = "Leader",
  llm_object = openai_4_1_mini
)

lead_agent$register_agents(c(stylist, stylist2))

lead_agent$judge_and_choose_best_response("what's the best way to war a kalvin klein shirt?")


## End(Not run)
```

Index