Package 'mini007'

September 5, 2025

Type Package

Title Lightweight Framework for Orchestrating Multi-Agent Large Language Models

Version 0.1.0

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), R6 (>= 2.6.1), uuid (>= 1.2.0)

RoxygenNote 7.3.2

Suggests ellmer

NeedsCompilation no

Author Mohamed El Fodil Ihaddaden [aut, cre]

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

Repository CRAN

Date/Publication 2025-09-05 11:40:08 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.

`messages` A list of past messages (system, user, assistant).

`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.

Methods

Public methods:

- `Agent$new()`
- `Agent$invoke()`
- `Agent$clone()`

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

Usage:

```
Agent$new(name, instruction, llm_object)
```

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)

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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)
```

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

Methods

Public methods:

- LeadAgent$new()
- LeadAgent$clear_agents()
- LeadAgent$remove_agents()
- LeadAgent$register_agents()
- LeadAgent$delegate_prompt()
- LeadAgent$invoke()
- LeadAgent$generate_plan()
- LeadAgent$broadcast()
- LeadAgent$set_hitl()
- LeadAgent$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 `delegate_prompt()`**:** Returns a list of subtasks with assigned agent IDs and names.

Usage:

```
LeadAgent$delegate_prompt(prompt)
```

Arguments:

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

Returns: A list of lists, each with 'agent_id', 'agent_name', and 'prompt' fields.

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

Usage:

```
LeadAgent$invoke(prompt)
```

Arguments:

`prompt` The complex user instruction to process.

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 `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)
```

Index