Package: LLMAgentR (via r-universe)

May 15, 2026

Type Package

Title Language Model Agents in R for AI Workflows and Research

Version 0.3.2

Maintainer Kwadwo Daddy Nyame Owusu Boakye

<kwadwo.owusuboakye@outlook.com>

Description Provides modular, graph-based agents powered by large language models (LLMs) for intelligent task execution in R. Supports structured workflows for tasks such as forecasting, data visualization, feature engineering, data wrangling, data cleaning, 'SQL', code generation, weather reporting, and research-driven question answering. Each agent performs iterative reasoning: recommending steps, generating R code, executing, debugging, and explaining results. Includes built-in support for packages such as 'tidymodels', 'modeltime', 'plotly', 'ggplot2', and 'prophet'. Designed for analysts, developers, and teams building intelligent, reproducible AI workflows in R. Compatible with LLM providers such as 'OpenAI', 'Anthropic', 'Groq', and 'Ollama'. Inspired by the Python package 'langagent'.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

URL https://github.com/knowusuboaky/LLMAgentR, https://knowusuboaky.github.io/LLMAgentR/

BugReports https://github.com/knowusuboaky/LLMAgentR/issues

Depends R (>= 4.1.0)

Imports plotly, stats, utils, DBI, RSQLite, dplyr, glue, httr, officer, purrr, timetk, pdftools, parsnip, recipes, workflows, rsample, modeltime.ensemble, modeltime, xml2

Suggests testthat (>= 3.0.0), roxygen2, knitr, rmarkdown, jsonlite, magrittr, rlang, tidyr, ggplot2, usethis, prophet, forcats, kernlab, xgboost, xfun, modeltime.resample, tidymodels, tibble, lubridate, methods, tesseract, rvest, fastDummies, stringr

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libpoppler-cpp-dev poppler-data libnode-dev libx11-dev

Repository https://knowusuboaky.r-universe.dev

Date/Publication 2026-02-14 19:55:00 UTC

RemoteUrl https://github.com/knowusuboaky/llmagentr

RemoteRef HEAD

RemoteSha fe20d91a4933f54e89150822857fcccеceb848d9

Contents

`as_mermaid` *Convert a Custom Graph Spec to Mermaid*

Description

Convert a Custom Graph Spec to Mermaid

Usage

```
as_mermaid(
  x,
  direction = c("TD", "LR"),
  subgraphs = NULL,
  include_start_end = TRUE,
  style = TRUE
)
```

Arguments

x	Graph spec list (from ‘build_custom_agent(..., output = "both")$graph‘) or a compiled object returned by ‘build_custom_agent(..., output = "both")‘ or [compile_graph()].
direction	Mermaid direction: ‘"TD"‘ (top-down) or ‘"LR"‘ (left-right).
subgraphs	Optional named list of subgraph groupings.
include_start_end	Logical; include ‘__start__‘ and ‘__end__‘ nodes.
style	Logical; include default LangGraph-like Mermaid styling.

Value

Mermaid flowchart text.

---

build_code_agent *Build an R Code-Generation Agent*

Description

Constructs an LLM-powered agent for generating, debugging, explaining, or optimizing R code. **Two calling patterns are supported**:

- **Builder pattern** – omit ‘user_input‘. The function returns a reusable **coder-agent closure**. Call that closure with different queries whenever you need code help.
- **One-shot pattern** – provide ‘user_input‘. The function executes immediately and returns the result once.

Arguments

llm	A function that accepts a character ‘prompt‘ and returns an LLM response (optionally accepts ‘verbose‘).
system_prompt	Optional system-level instructions that override the built-in default prompt.
user_input	The coding task/query (e.g., ‘"Write function to filter NAs"‘). **Default ‘NULL‘** – omit to obtain a reusable agent.
max_tries	Maximum LLM retry attempts (default ‘3‘).
backoff	Seconds to wait between retries (default ‘2‘).
verbose	Logical flag to show progress messages (default ‘TRUE‘).

Details

The agent automatically retries failed LLM calls (with exponential back-off) and always returns a structured result.

Value

- If ‘user_input‘ is ‘NULL‘: a **function** (the coder-agent closure).
- Otherwise: a **list** with the fields

input The original user query.

llm_response The LLM output (or error message).

system_prompt Prompt actually sent.

success Logical; did the call succeed?

attempts Number of attempts made.

Examples

```
## Not run:
## ------------------------------------------------------------------
## 1)  Builder pattern – create a reusable coder agent
## ------------------------------------------------------------------
coder <- build_code_agent(
  llm       = my_llm_wrapper,   # your own wrapper around the LLM API
  max_tries = 3,
  backoff   = 2,
  verbose   = FALSE
)

# Use the agent multiple times
res1 <- coder("Write an R function that z-score-standardises all numeric columns.")
res2 <- coder("Explain what `%>%` does in tidyverse pipelines.")

## ------------------------------------------------------------------
## 2)  One-shot pattern – run a single request immediately
## ------------------------------------------------------------------
one_shot <- build_code_agent(
  llm        = my_llm_wrapper,
  user_input = "Create a ggplot2 bar chart of mpg by cyl in mtcars.",
  max_tries  = 3,
  backoff    = 2,
  verbose    = FALSE
)

## End(Not run)
```

build_custom_agent *Build a Custom Graph-Based Agent*

Description

Build a reusable agent by wiring user-defined node functions into a state graph. This is the public extension point for creating custom LLMAgentR workflows.

Usage

```
build_custom_agent(
  node_functions,
  entry_point,
  edges = list(),
  conditional_edges = list(),
  default_state = list(),
  checkpointer = NULL,
  output = c("agent", "mermaid", "both"),
  direction = c("TD", "LR"),
  subgraphs = NULL,
  style = TRUE
)
```

Arguments

node_functions	Named list of node functions. Each function takes ‘state‘ and returns a named list or [make_command()].
entry_point	Name of the start node (must exist in ‘node_functions‘).
edges	Optional list of static edges. Each item can be: - a character vector of length 2: ‘c("from", "to")‘, or - a list with ‘from‘ and ‘to‘.
conditional_edges	Optional list where each item contains: - ‘from‘ (or ‘node_name‘), - ‘condition‘ (or ‘condition_fun‘), - ‘mapping‘ (or ‘mapping_list‘) as a named list of label -> node.
default_state	Optional named list merged into ‘state‘ for missing keys.
checkpointer	Optional callback ‘function(state, current_node)‘ executed after each transition.
output	Output mode: - ‘"agent"‘ (default): return runnable agent function, - ‘"mermaid"‘: return Mermaid text only, - ‘"both"‘: return list with ‘run‘, ‘graph‘, and ‘mermaid‘.
direction	Mermaid direction used when ‘output‘ includes Mermaid.
subgraphs	Optional named list of subgraph groups for Mermaid rendering.
style	Logical; include default Mermaid class styling.

Value

- If ‘output = "agent"‘: a function that accepts ‘state‘ and returns final state. - If ‘output = "mermaid"‘: Mermaid flowchart text. - If ‘output = "both"‘: list with ‘run‘, ‘graph‘, and ‘mermaid‘.

Examples

```
## Not run:
custom <- build_custom_agent(
  node_functions = list(
    start = function(state) make_command("classify"),
    classify = function(state) {
      if (grepl("weather", state$query, ignore.case = TRUE)) {
        make_command("weather")
      } else {
        make_command("general")
      }
    },
    weather = function(state) list(answer = "Routing to weather handler"),
    general = function(state) list(answer = "Routing to general handler")
  ),
  entry_point = "start",
  edges = list(c("weather", "__end__"), c("general", "__end__")),
  output = "both",
  subgraphs = list(
    Router = c("start", "classify"),
    Handlers = c("weather", "general")
  )
)

cat(custom$mermaid)
custom$run(list(query = "weather in Accra"))

## End(Not run)
```

---

build_custom_multi_agent

Build a Custom Multi-Agent Team (Supervisor Style)

Description

Build a supervisor-routed multi-agent workflow, similar to LangGraph team orchestration. The supervisor chooses the next worker by setting ‘state$next‘ to one of the worker names or ‘FINISH‘.

Usage

```
build_custom_multi_agent(
  supervisor,
  workers,
```

```
  finish_token = "FINISH",
  max_turns = 10L,
  allow_repeat = FALSE,
  worker_error_policy = c("return_to_supervisor", "stop"),
  default_state = list(),
  checkpointer = NULL,
  output = c("agent", "mermaid", "both"),
  direction = c("TD", "LR"),
  subgraphs = NULL,
  style = TRUE
)
```

Arguments

supervisor	Function that accepts ‘state‘ and returns: - worker name (character scalar), or - list with ‘next‘, optionally additional updates.
workers	Named list of workers. Each worker can be: - a function that accepts ‘state‘ and returns a named list of updates, or - a compiled custom-agent object with a callable ‘$run‘ function (for example from [build_custom_agent()] with ‘output = "both"‘ or [compile_graph()]).
finish_token	Character label used by the supervisor to terminate.
max_turns	Maximum worker turns before forcing finish.
allow_repeat	Logical; allow the same worker twice in a row.
worker_error_policy	‘"return_to_supervisor"‘ (default) or ‘"stop"‘.
default_state	Optional defaults merged into incoming ‘state‘.
checkpointer	Optional callback ‘function(state, current_node)‘.
output	Output mode: - ‘"agent"‘ (default): return runnable agent, - ‘"mermaid"‘: return Mermaid text, - ‘"both"‘: return list with ‘run‘, ‘graph‘, and ‘mermaid‘.
direction	Mermaid direction when ‘output‘ includes Mermaid.
subgraphs	Optional Mermaid subgraph groups. If ‘NULL‘, defaults to ‘list(Supervisor = "supervisor", Workers = names(workers))‘.
style	Logical; include default Mermaid class styling.

Value

- If ‘output = "agent"‘: runnable function(state). - If ‘output = "mermaid"‘: Mermaid text. - If ‘output = "both"‘: list with ‘run‘, ‘graph‘, and ‘mermaid‘.

Examples

```
## Not run:
supervisor_fn <- function(state) {
  if (is.null(state$turn) || state$turn == 0) "Researcher" else "FINISH"
}
```

```
workers <- list(
  Researcher = function(state) {
    list(result = "Research complete")
  },
  Writer = function(state) {
    list(result = "Draft complete")
  }
)

team <- build_custom_multi_agent(
  supervisor = supervisor_fn,
  workers = workers,
  output = "both"
)

cat(team$mermaid)
team$run(list())

## End(Not run)
```

`build_data_cleaning_agent` *Build a Data Cleaning Agent*

Description

Constructs a multi-step agent workflow to recommend, generate, fix, execute, and explain robust R code for data cleaning tasks using LLMs and user-defined data.

Arguments

`model`	A function that accepts a prompt and returns a text response (e.g., OpenAI, Claude).
`data_raw`	A raw data.frame (or list convertible to data.frame) to be cleaned.
`human_validation`	Logical; whether to include a manual review step.
`bypass_recommended_steps`	Logical; whether to skip LLM-based cleaning step suggestions.
`bypass_explain_code`	Logical; whether to skip explanation of the generated code.
`verbose`	Logical; whether to print progress messages (default: TRUE)

Value

A compiled graph-based cleaning agent function that accepts and mutates a state list.

Examples

```
## Not run:
# 1) Load the data
data <- read.csv("tests/testthat/test-data/churn_data.csv")

# 2) Create the agent
data_cleaner_agent <- build_data_cleaning_agent(
  model = my_llm_wrapper,
  human_validation = FALSE,
  bypass_recommended_steps = FALSE,
  bypass_explain_code = FALSE,
  verbose = FALSE
)

# 3) Define the initial state
initial_state <- list(
  data_raw = data,
  user_instructions = "Don't remove outliers when cleaning the data.",
  max_retries = 3,
  retry_count = 0
)

# 4) Run the agent
final_state <- data_cleaner_agent(initial_state)

## End(Not run)
```

`build_data_wrangling_agent`

Build a Data Wrangling Agent

Description

Constructs a state graph-based agent that recommends, generates, executes, fixes, and explains data wrangling transformations based on user instructions and dataset structure. The resulting function handles list or single data frame inputs and produces a cleaned dataset.

Arguments

`model`	A function that takes a prompt string and returns LLM-generated output.
`human_validation`	Logical; whether to enable manual review step before code execution.
`bypass_recommended_steps`	Logical; skip initial recommendation of wrangling steps.
`bypass_explain_code`	Logical; skip final explanation step after wrangling.
`verbose`	Logical; whether to print progress messages (default: TRUE)

Value

A callable agent function that mutates a provided ‘state‘ list by populating: - ‘data_wrangled‘: the final cleaned data frame, - ‘data_wrangler_function‘: the code used, - ‘data_wrangler_error‘: any execution error (if occurred), - ‘wrangling_report‘: LLM-generated explanation (if ‘bypass_explain_code = FALSE‘)

Examples

```
## Not run:
# 1) Simulate multiple data frames with a common ID
df1 <- data.frame(
  ID = c(1, 2, 3, 4),
  Name = c("John", "Jane", "Jim", "Jill"),
  stringsAsFactors = FALSE
)

df2 <- data.frame(
  ID = c(1, 2, 3, 4),
  Age = c(25, 30, 35, 40),
  stringsAsFactors = FALSE
)

df3 <- data.frame(
  ID = c(1, 2, 3, 4),
  Education = c("Bachelors", "Masters", "PhD", "MBA"),
  stringsAsFactors = FALSE
)

# 2) Combine into a list
data <- list(df1, df2, df3)

# 3) Create the agent
data_wrangling_agent <- build_data_wrangling_agent(
  model = my_llm_wrapper,
  human_validation = FALSE,
  bypass_recommended_steps = FALSE,
  bypass_explain_code = FALSE,
  verbose = FALSE
)

# 4) Define the initial state
initial_state <- list(
  data_raw = data,
  user_instructions = "Merge the data frames on the ID column.",
  max_retries = 3,
  retry_count = 0
)

# 5) Run the agent
final_state <- data_wrangling

## End(Not run)
```

build_doc_summarizer_agent
Build a Document Summarizer Agent

Description

Creates an LLM-powered document summarization workflow that processes PDF, DOCX, PPTX, TXT, or plain text input and returns structured markdown summaries.

Usage

```
build_doc_summarizer_agent(
  llm,
  summary_template = NULL,
  chunk_size = 4000,
  overlap = 200,
  verbose = TRUE,
  output = c("agent", "mermaid", "both"),
  direction = c("TD", "LR"),
  subgraphs = NULL,
  style = TRUE
)
```

Arguments

llm	A function that accepts a character prompt and returns an LLM response.
summary_template	Optional custom summary template in markdown format.
chunk_size	Maximum character length for document chunks (default: 4000).
overlap	Character overlap between chunks (default: 200).
verbose	Logical controlling progress messages (default: TRUE).
output	Output type: "agent" (default), "mermaid" for diagram only, or "both".
direction	Mermaid diagram direction: "TD" (top-down) or "LR" (left-right).
subgraphs	Optional named list for grouping nodes in Mermaid diagram.
style	Logical; apply default styling to Mermaid diagram (default: TRUE).

Value

A function that accepts file paths or text input and returns:

- summary - The generated markdown summary
- metadata - Document metadata if available
- chunks - Number of processing chunks used
- success - Logical indicating success

Examples

```
## Not run:
# Build document summarizer agent
summarizer_agent <- build_doc_summarizer_agent(
  llm = my_llm_wrapper,
  summary_template = NULL,
  chunk_size = 4000,
  overlap = 200,
  verbose = FALSE
)

# Summarize document
final_state <- summarizer_agent("https://github.com/knowusuboaky/LLMAgentR/raw/main/\
tests/testthat/test-data/scrum.docx")

## End(Not run)
```

`build_feature_engineering_agent`
Build a Feature Engineering Agent

Description

Constructs a graph-based feature engineering agent that guides the process of: recommending, generating, executing, fixing, and explaining feature engineering code.

Arguments

`model`	A function that accepts a prompt and returns an LLM-generated response.
`human_validation`	Logical; include a manual review node before code execution.
`bypass_recommended_steps`	Logical; skip the LLM-based recommendation phase.
`bypass_explain_code`	Logical; skip final explanation step.
`verbose`	Logical; whether to print progress messages (default: TRUE)

Value

A callable agent function that executes feature engineering via a state graph.

Examples

```
## Not run:
# 1) Load the data
data <- read.csv("tests/testthat/test-data/churn_data.csv")
```

```
# 2) Create the feature engineering agent
feature_engineering_agent <- build_feature_engineering_agent(
  model = my_llm_wrapper,
  human_validation = FALSE,
  bypass_recommended_steps = FALSE,
  bypass_explain_code = FALSE,
  verbose = TRUE
)

# 3) Define the initial state
initial_state <- list(
  data_raw = data,
  target_variable = "Churn",
  user_instructions = "Inspect the data. Make any new features and transformations
  that you think will be useful for predicting the target variable.",
  max_retries = 3,
  retry_count = 0
)

# 4) Run the agent
final_state <- feature_engineering_agent(initial_state)

## End(Not run)
```

`build_forecasting_agent` *Build a Time Series Forecasting Agent*

Description

Constructs a state graph-based forecasting agent that: recommends forecasting steps, extracts parameters, generates code, executes the forecast using ‘modeltime‘, fixes errors if needed, and explains the result. It leverages multiple models including Prophet, XGBoost, Random Forest, SVM, and Prophet Boost, and combines them in an ensemble.

Arguments

`model`	A function that takes a prompt and returns an LLM-generated result.
`bypass_recommended_steps`	Logical; skip initial step recommendation.
`bypass_explain_code`	Logical; skip the final explanation step.
`mode`	Visualization mode for forecast plots. One of ‘"light"‘ or ‘"dark"‘.
`line_width`	Line width used in plotly forecast visualization.
`verbose`	Logical; whether to print progress messages.

Value

A callable agent function that mutates the given ‘state‘ list.

Examples

```
## Not run:
# 2) Prepare the dataset
my_data <- walmart_sales_weekly

# 3) Create the forecasting agent
forecasting_agent <- build_forecasting_agent(
  model = my_llm_wrapper,
  bypass_recommended_steps = FALSE,
  bypass_explain_code = FALSE,
  mode = "dark", # dark or light
  line_width = 3,
  verbose = FALSE
)

# 4) Define the initial state
initial_state <- list(
 user_instructions = "Forecast sales for the next 30 days, using `id` as the grouping variable,
  a forecasting horizon of 30, and a confidence level of 90%.",
  data_raw = my_data
)

# 5) Run the agent
final_state <- forecasting_agent(initial_state)

## End(Not run)
```

---

build_interpreter_agent

Build an Interpreter Agent

Description

Constructs an LLM-powered agent that explains plots, tables, text, or other outputs for both technical and non-technical audiences.

Arguments

`llm`	Function that takes `prompt` and returns an LLM response (may or may not accept `verbose`).
`interpreter_prompt`	Optional template for the prompt (default supplied).
`code_output`	The output to interpret (plot caption, table text, model summary, etc.). **Default `NULL`**.

`max_tries`	Max LLM retry attempts (default `3`).
`backoff`	Seconds between retries (default `2`).
`verbose`	Logical; print progress (default `TRUE`).

Details

Two calling patterns

- **Builder pattern** – omit `code_output`; a reusable **interpreter-agent closure** is returned.
- **One-shot pattern** – provide `code_output`; the function runs immediately and returns the interpretation.

Value

- If `code_output` is `NULL`: a **function** (closure).
- Otherwise: a **list** with

  prompt The full prompt sent to the LLM.

  interpretation The LLM's explanation (or error).

  success Logical; did it succeed?

  attempts Number of attempts made.

Examples

```
## Not run:
## 1) Builder pattern -------------------------------------------
interp <- build_interpreter_agent(llm = my_llm_wrapper, verbose = FALSE)

table_txt <- "
| Region | Sales | Profit |
| North  | 2000  | 300    |
| South  | 1500  | 250    |"

res1 <- interp(table_txt)
res2 <- interp("R² = 0.87 for the fitted model . . . ")

## 2) One-shot pattern ------------------------------------------
build_interpreter_agent(
  llm         = my_llm_wrapper,
  code_output = table_txt,
  verbose     = FALSE
)

## End(Not run)
```

`build_researcher_agent` *Build a Web Researcher Agent*

Description

Constructs an LLM-powered research agent that performs web searches (via Tavily API) and generates structured responses based on search results. The agent handles different question types (general knowledge, comparisons, controversial topics) with appropriate response formats.

Arguments

`llm`	A function that accepts a character prompt and returns an LLM response. (It must accept ‘prompt‘ and optionally ‘verbose‘.)
`tavily_search`	Tavily API key as a string or NULL to use ‘Sys.getenv("TAVILY_API_KEY")‘.
`system_prompt`	Optional custom system prompt for the researcher agent.
`max_results`	Number of web search results to retrieve per query (default: 5).
`max_tries`	Maximum number of retry attempts for search or LLM call (default: 3).
`backoff`	Initial wait time in seconds between retries (default: 2).
`verbose`	Logical flag to control progress messages (default: TRUE).

Value

A function that accepts a user query string and returns a list with:

- query - The original research query.
- prompt - The full prompt sent to the LLM.
- response - The generated LLM response.
- search_results - Raw search results (if any were found).
- success - Logical indicating if research succeeded (both search and LLM).

Examples

```
## Not run:
# Initialize researcher agent
researcher_agent <- build_researcher_agent(
  llm = my_llm_wrapper,
  tavily_search = NULL,
  system_prompt = NULL,
  max_results = 5,
  max_tries = 3,
  backoff = 2,
  verbose = FALSE
)
```

```
# Perform research
result <- researcher_agent("Who is Messi?")

## End(Not run)
```

`build_sql_agent` *Build a SQL Agent Graph*

Description

This function constructs a full SQL database agent using a graph-based workflow. It supports step recommendation, SQL code generation, error handling, optional human review, and automatic explanation of the final code.

Arguments

`model`	A function that accepts prompts and returns LLM responses.
`connection`	A DBI connection object to the target SQL database.
`n_samples`	Number of candidate SQL plans to consider (used in prompt).
`human_validation`	Whether to include a human review node.
`bypass_recommended_steps`	If TRUE, skip the step recommendation node.
`bypass_explain_code`	If TRUE, skip the final explanation step.
`verbose`	Logical indicating whether to print progress messages (default: TRUE).

Value

A compiled SQL agent function that runs via a state machine (graph execution).

Examples

```
## Not run:
# 1) Connect to the database
conn <- DBI::dbConnect(RSQLite::SQLite(), "tests/testthat/test-data/northwind.db")

# 2) Create the SQL agent
sql_agent <- build_sql_agent(
  model                    = my_llm_wrapper,
  connection               = conn,
  human_validation         = FALSE,
  bypass_recommended_steps = FALSE,
  bypass_explain_code      = FALSE,
  verbose                  = FALSE
)
```

```
# 3) Define the initial state
initial_state <- list(
  user_instructions = "Identify the Regions (or Territories) with the highest
  CustomerCount and TotalSales.
  Return a table with columns: Region, CustomerCount, and TotalSales.
Hint: (UnitPrice × Quantity).",
  max_retries       = 3,
  retry_count       = 0
)

# 4) Run the agent
final_state <- sql_agent(initial_state)

## End(Not run)
```

`build_visualization_agent` *Build Visualization Agent*

Description

Creates a data visualization agent with configurable workflow steps.

Arguments

`model`	The AI model function to use for code generation
`human_validation`	Whether to include human validation step (default: FALSE)
`bypass_recommended_steps`	Skip recommendation step (default: FALSE)
`bypass_explain_code`	Skip explanation step (default: FALSE)
`function_name`	Name for generated visualization function (default: "data_visualization")
`verbose`	Whether to print progress messages (default: TRUE)

Value

A function that takes state and returns visualization results

Examples

```
## Not run:
# 1) Load the data
data <- read.csv("tests/testthat/test-data/churn_data.csv")

# 2) Create the visualization agent
visualization_agent <- build_visualization_agent(
```

```
  model = my_llm_wrapper,
  human_validation = FALSE,
  bypass_recommended_steps = FALSE,
  bypass_explain_code = FALSE,
  verbose = FALSE
)

# 3) Define the initial state
initial_state <- list(
  data_raw = data,
  target_variable = "Churn",
  user_instructions = "Create a clean and visually appealing box plot to show
  the distribution of Monthly Charges across Churn categories.
  Use distinct colors for each Churn group,
  add clear axis labels, a legend, and a meaningful title.",
  max_retries = 3,
  retry_count = 0
)

# 4) Run the agent
final_state <- visualization_agent(initial_state)

## End(Not run)
```

`build_weather_agent` *Build a Weather Agent*

Description

Constructs an LLM-powered weather assistant that fetches data from OpenWeatherMap and generates user-friendly reports. Handles location parsing, API calls, caching, and LLM-based summarization.

Arguments

`llm`	A function that accepts a character prompt and returns an LLM response.
`location_query`	Free-text location query (e.g., "weather in Toronto").
`system_prompt`	Optional LLM system prompt for weather reporting.
`weather_api_key`	OpenWeatherMap API key (defaults to OPENWEATHERMAP_API_KEY env var).
`units`	Unit system ("metric" or "imperial").
`n_tries`	Number of retry attempts for API/LLM calls (default: 3).
`backoff`	Base seconds to wait between retries (default: 2).
`endpoint_url`	OpenWeatherMap endpoint URL.
`verbose`	Logical controlling progress messages (default: TRUE).

Value

A list containing:

- success - Logical indicating if operation succeeded
- location - Cleaned location string
- weather_raw - Raw API response
- weather_formatted - Formatted weather string
- llm_response - Generated weather report
- timestamp - Time of response
- cache_hit - Logical indicating cache usage
- attempts - Number of tries made

Examples

```
## Not run:
# Get weather information
weather_agent <- build_weather_agent(
  llm = my_llm_wrapper,
  location_query = "Tokyo, Japan",
  system_prompt = NULL,
  weather_api_key = NULL,
  units = "metric", # metric or imperial
  n_tries = 3,
  backoff = 2,
  endpoint_url = NULL,
  verbose = FALSE
)

## End(Not run)
```

---

compile_graph *Compile a Custom Agent Graph (LangGraph-Style Output)*

Description

Convenience wrapper around [build_custom_agent()] that returns both runnable agent and Mermaid graph artifacts.

Usage

```
compile_graph(
  node_functions,
  entry_point,
  edges = list(),
  conditional_edges = list(),
  default_state = list(),
```

```
  checkpointer = NULL,
  direction = c("TD", "LR"),
  subgraphs = NULL,
  style = TRUE
)
```

Arguments

node_functions	Named list of node functions. Each function takes ‘state‘ and returns a named list or [make_command()].
entry_point	Name of the start node (must exist in ‘node_functions‘).
edges	Optional list of static edges. Each item can be: - a character vector of length 2: ‘c("from", "to")‘, or - a list with ‘from‘ and ‘to‘.
conditional_edges	Optional list where each item contains: - ‘from‘ (or ‘node_name‘), - ‘condition‘ (or ‘condition_fun‘), - ‘mapping‘ (or ‘mapping_list‘) as a named list of label -> node.
default_state	Optional named list merged into ‘state‘ for missing keys.
checkpointer	Optional callback ‘function(state, current_node)‘ executed after each transition.
direction	Mermaid direction used when ‘output‘ includes Mermaid.
subgraphs	Optional named list of subgraph groups for Mermaid rendering.
style	Logical; include default Mermaid class styling.

Value

A list with ‘run‘, ‘graph‘, and ‘mermaid‘.

---

save_mermaid_png *Save Mermaid Diagram as PNG*

Description

Render Mermaid text (or a compiled graph object) to a PNG file using Mermaid CLI (‘mmdc‘).

Usage

```
save_mermaid_png(
  x,
  file,
  mmdc = Sys.which("mmdc"),
  direction = c("TD", "LR"),
  subgraphs = NULL,
  include_start_end = TRUE,
  style = TRUE,
  width = NULL,
```

```
  height = NULL,
  scale = NULL,
  background = "white",
  theme = "default",
  quiet = TRUE
)
```

Arguments

x	Mermaid text, graph spec, or compiled object returned by [build_custom_agent()] with ‘output = "both"‘ or [compile_graph()].
file	Output ‘.png‘ path.
mmdc	Path to Mermaid CLI executable. Defaults to ‘Sys.which("mmdc")‘.
direction	Mermaid direction used when ‘x‘ is not plain Mermaid text.
subgraphs	Optional named list of subgraph groupings.
include_start_end	Logical; include ‘__start__‘ and ‘__end__‘ nodes.
style	Logical; include default Mermaid class styling.
width	Optional diagram width passed to ‘mmdc‘.
height	Optional diagram height passed to ‘mmdc‘.
scale	Optional diagram scale passed to ‘mmdc‘.
background	Background color for Mermaid rendering.
theme	Mermaid theme (for example ‘"default"‘, ‘"neutral"‘, ‘"dark"‘).
quiet	Logical; suppress Mermaid CLI output when ‘TRUE‘.

Value

Invisibly returns the output file path.

---

state_graph_utils *State Graph Utilities for Custom Agents*

Description

Lightweight graph primitives used by LLMAgentR's workflow agents. These utilities are exported so users can build custom state-machine agents.

Usage

```
make_node(func, name = NULL)

make_edge(from, to, condition = NULL, label = NULL)

make_command(goto = NULL, update = list())

interrupt(value)

StateGraph()
```

Arguments

func	A function that accepts a ‘state‘ list and returns either: 1) a named list of state updates, or 2) a command list created by [make_command()].
name	Optional node name label.
from	Source node name.
to	Destination node name.
condition	Optional function ‘function(state)‘ that returns a label used for conditional routing.
label	Optional label matched against the value returned by ‘condition‘.
goto	Next node name to jump to.
update	Named list of state fields to merge before jumping.
value	Prompt text shown to the user.

Value

A list with ‘func‘ and ‘name‘.

A list with ‘from‘, ‘to‘, ‘condition‘, and ‘label‘.

A command-like list with ‘goto‘ and ‘update‘.

A character string from ‘readline()‘.

A list with methods: - ‘add_node(name, func)‘ - ‘add_edge(from, to)‘ - ‘add_conditional_edges(node_name, condition_fun, mapping_list)‘ - ‘set_entry_point(node_name)‘ - ‘compile(checkpointer = NULL)‘ - ‘END_NODE_NAME‘

Index