Package ‘OmixBenchR’

October 15, 2025

Type Package

Title Benchmarking Large Language Models in Omics Analysis

Version 1.0.0

Description Benchmarking of Large Language Models for Multi-omics Analysis

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Depends R (>= 2.10)

Imports llmhelper

Contents

`bioinfo_prompt_formatter` *Generate standardized bioinformatics analysis prompt from existing materials*

Description

This function takes an existing prompt, expert code, and expected answer to generate a standardized prompt following the bioinformatics analysis task format.

Usage

```
bioinfo_prompt_formatter(current_prompt, gold_code, answer, llm_client)
```

Arguments

`current_prompt`	Character string. The original/current prompt text
`gold_code`	Character string. The expert-written code that correctly solves the task
`answer`	Character string. The expected answer (used for reference but not revealed in output)
`llm_client`	LLM provider object created by llm_openai() or llm_ollama().

Value

Character string containing the standardized prompt

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
## Not run:
# Example usage
standardized_prompt <- bioinfo_prompt_formatter(
  current_prompt = "Find the most upregulated gene...",
  gold_code = "library(limma)\n...",
  answer = "GENE1"
)

## End(Not run)
```

`CodeEval` *Evaluate Bioinformatics R Code Using LLM*

Description

This function evaluates bioinformatics R code solutions using a Language Learning Model (LLM) based on scientific correctness, technical quality, and professional standards. It returns a structured JSON assessment with detailed scoring across multiple dimensions.

Usage

```
CodeEval(
  task,
  response,
  llm_client,
  max_retries = 3,
  schema_strict = TRUE,
  schema_type = "auto",
  verbose = FALSE
)
```

Arguments

`task`	A character string containing the bioinformatics task or question that the code is supposed to solve. This provides context for the evaluation and helps the LLM assess whether the code addresses the biological question appropriately.
`response`	A character string containing the R code to be evaluated. This should be the complete code solution that attempts to solve the given bioinformatics task.
`llm_client`	An LLM provider object created by functions like `llm_openai()` or `llm_ollama()`. This object contains the configuration for connecting to and communicating with the specific LLM service that will perform the evaluation.
`max_retries`	Integer. Maximum number of retry attempts if the LLM fails to provide a valid JSON response (default: 3). The function will retry if JSON parsing fails or if the response doesn't match the expected schema structure.
`schema_strict`	Logical. Whether to enforce strict JSON schema validation (default: TRUE). When TRUE, the LLM response must exactly match the predefined evaluation schema with no additional properties allowed. When FALSE, allows more flexible JSON structure.
`schema_type`	Character. Method for enforcing JSON response format (default: "auto"). Options include: • "auto": Automatically detect best method based on LLM provider • "text-based": Add JSON instructions to prompt (works with any provider) • "openai": Use OpenAI's native JSON mode (requires compatible OpenAI API) • "ollama": Use Ollama's native JSON mode (requires compatible Ollama model) • "openai_oo": OpenAI mode without schema enforcement in API • "ollama_oo": Ollama mode without schema enforcement in API
`verbose`	Logical. Whether to print detailed interaction logs to console (default: FALSE). When TRUE, shows the prompt being sent, the LLM's response, and any retry attempts. Useful for debugging and monitoring the evaluation process.

Details

This function implements a comprehensive evaluation framework specifically designed for bioinformatics code assessment. The evaluation covers three main areas:

Core Scientific Validity (45 points):

- Problem solving approach and biological question addressing
- Technical implementation with appropriate bioinformatics methods
- Complete analysis workflows with logical flow

Technical Quality (30 points):

- Data handling including preprocessing and quality control
- Statistical rigor with proper corrections and methods

Professional Excellence (25 points):

- Domain knowledge and biological interpretation
- Code robustness and error handling

- Documentation and usability

The function uses a positive scoring system where points are awarded for good practices rather than deducted for shortcomings. This encourages comprehensive evaluation and recognizes partial solutions that demonstrate scientific understanding.

The evaluation leverages the `llmhelper::get_llm_response()` function with JSON schema enforcement to ensure consistent, structured output that can be easily processed and analyzed programmatically.

Value

A named list containing the evaluation results, or a list with an error message if evaluation fails. The successful return structure includes:

- `total_score`: Integer (0-100) representing the overall evaluation score
- `breakdown`: Named list with individual dimension scores:
  - `problem_solving`: Score 0-20 for addressing the biological question
  - `technical_implementation`: Score 0-25 for appropriate methods and workflow
  - `data_handling`: Score 0-15 for preprocessing and quality control
  - `statistical_rigor`: Score 0-15 for statistical methods and corrections
  - `domain_knowledge`: Score 0-10 for biological context understanding
  - `robustness`: Score 0-10 for error handling and validation
  - `documentation`: Score 0-5 for code clarity and documentation

If evaluation fails, returns `list(error = "error message")`.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

See Also

`get_llm_response` for the underlying LLM communication function, `llm_openai` for creating OpenAI-compatible LLM providers, `llm_ollama` for creating Ollama LLM providers

Examples

```
## Not run:
# Set up LLM client
library(llmhelper)
client <- llm_openai(
  base_url = "https://api.openai.com/v1/chat/completions",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  model = "gpt-4",
  temperature = 0.2
)

# Define a bioinformatics task
task <- "Analyze RNA-seq data to identify differentially expressed genes"

# Example R code to evaluate
code <- "
library(DESeq2)
library(ggplot2)
```

```
# Load count data
counts <- read.csv('counts.csv', row.names=1)
coldata <- read.csv('coldata.csv', row.names=1)

# Create DESeq2 object
dds <- DESeqDataSetFromMatrix(countData = counts,
                              colData = coldata,
                              design = ~ condition)

# Run DESeq2 analysis
dds <- DESeq(dds)
res <- results(dds)

# Apply multiple testing correction
res$padj <- p.adjust(res$pvalue, method='BH')

# Filter significant genes
sig_genes <- subset(res, padj < 0.05 & abs(log2FoldChange) > 1)
print(summary(res))
"

# Evaluate the code
evaluation <- CodeEval(task, code, client, verbose = TRUE)

# Print results
cat("Total Score:", evaluation$total_score, "/100\n")
cat("Problem Solving:", evaluation$breakdown$problem_solving, "/20\n")
cat("Technical Implementation:", evaluation$breakdown$technical_implementation, "/25\n")

# Batch evaluation example
tasks <- c("Perform GO enrichment analysis", "Create a volcano plot")
codes <- c("library(clusterProfiler)...", "library(ggplot2)...")

results <- mapply(CodeEval, tasks, codes,
  MoreArgs = list(llm_client = client),
  SIMPLIFY = FALSE
)

## End(Not run)
```

`Execute_Task` *Execute Single Bioinformatics Task with Resume Capability (Dual Engine Support)*

Description

Execute Single Bioinformatics Task with Resume Capability (Dual Engine Support)

Usage

```
Execute_Task(
  task_prompt,
  task_name = NULL,
  llm_client,
  execution_engine = c("tidyprompt", "llmflow"),
  timeout_sec = 1200,
  max_interactions = 10,
  return_mode = NULL,
  save_file = NULL,
  force_rerun = FALSE,
  pkgs_to_use = c(),
  objects_to_use = list(),
  existing_session = NULL,
  list_packages = TRUE,
  list_objects = TRUE,
  return_session_info = TRUE,
  evaluate_code = TRUE,
  r_session_options = list()
)
```

Arguments

`task_prompt`	Character string containing the task prompt
`task_name`	Optional task name for logging (default: auto-generated)
`llm_client`	LLM client object. Type depends on execution_engine: • For "tidyprompt": LLM provider object • For "llmflow": Chat object from ellmer
`execution_engine`	Execution engine to use: "tidyprompt" or "llmflow" (default: "tidyprompt")
`timeout_sec`	Timeout in seconds (default: 1200)
`max_interactions`	Maximum LLM interactions (default: 10)
`return_mode`	Return mode specification: • For "tidyprompt": "full" or "only_response" • For "llmflow": "full", "code", "console", "object", "formatted_output", "llm_answer", "session"
`save_file`	Optional save file path for result
`force_rerun`	Force re-execution even if result exists (default: FALSE)
`pkgs_to_use`	(llmflow only) Packages to load in R session (default: c())
`objects_to_use`	(llmflow only) Named list of objects to load in R session (default: list())
`existing_session`	(llmflow only) Existing callr session to continue from (optional)
`list_packages`	(llmflow only) Whether to list available packages in prompt (default: TRUE)
`list_objects`	(llmflow only) Whether to list available objects in prompt (default: TRUE)
`return_session_info`	(llmflow only) Whether to return session state information (default: TRUE)
`evaluate_code`	(llmflow only) Whether to evaluate the generated code (default: TRUE)
`r_session_options`	(llmflow only) Options for callr R session (default: list())

Value

Task result object. Return structure depends on execution_engine and return_mode.

Examples

```
## Not run:
# === tidyprompt mode ===
result <- Execute_Task(
  task_prompt = "Analyze this data",
  llm_client = my_llm_provider,
  execution_engine = "tidyprompt",
  return_mode = "full"
)

# === llmflow mode ===
result <- Execute_Task(
  task_prompt = "Calculate mean of iris$Sepal.Length",
  llm_client = my_chat_obj,
  execution_engine = "llmflow",
  return_mode = "full",
  pkgs_to_use = c("dplyr"),
  objects_to_use = list(mydata = iris)
)

## End(Not run)
```

`Execute_Tasks` *Execute Multiple Bioinformatics Tasks with Resume Capability (Dual Engine Support)*

Description

Execute Multiple Bioinformatics Tasks with Resume Capability (Dual Engine Support)

Usage

```
Execute_Tasks(
  tasks,
  llm_client,
  execution_engine = c("tidyprompt", "llmflow"),
  timeout_sec = 1200,
  max_interactions = 10,
  return_mode = NULL,
  save_file = NULL,
  force_rerun = FALSE,
  pkgs_to_use = c(),
  objects_to_use = list(),
  existing_session = NULL,
  list_packages = TRUE,
  list_objects = TRUE,
  return_session_info = TRUE,
  evaluate_code = TRUE,
  r_session_options = list(),
  persist_session = FALSE
)
```

Arguments

`tasks`	Named list where each element is a task prompt string
`llm_client`	LLM client object. Type depends on execution_engine: • For "tidyprompt": LLM provider object • For "llmflow": Chat object from ellmer
`execution_engine`	Execution engine to use: "tidyprompt" or "llmflow" (default: "tidyprompt")
`timeout_sec`	Timeout in seconds (default: 1200)
`max_interactions`	Maximum LLM interactions (default: 10)
`return_mode`	Return mode specification: • For "tidyprompt": "full" or "only_response" • For "llmflow": "full", "code", "console", "object", "formatted_output", "llm_answer", "session" (default: "full" for both engines)
`save_file`	Custom save file path (optional, auto-generated if NULL)
`force_rerun`	Force re-execution of all tasks (default: FALSE)
`pkgs_to_use`	(llmflow only) Packages to load in R session (default: c())
`objects_to_use`	(llmflow only) Named list of objects to load in R session (default: list())
`existing_session`	(llmflow only) Existing callr session to continue from (optional)
`list_packages`	(llmflow only) Whether to list available packages in prompt (default: TRUE)
`list_objects`	(llmflow only) Whether to list available objects in prompt (default: TRUE)
`return_session_info`	(llmflow only) Whether to return session state information (default: TRUE)
`evaluate_code`	(llmflow only) Whether to evaluate the generated code (default: TRUE)
`r_session_options`	(llmflow only) Options for callr R session (default: list())
`persist_session`	(llmflow only) Whether to persist R session across tasks (default: FALSE)

Value

Named list of completed task results

Examples

```
## Not run:
# === tidyprompt mode ===
tasks <- list(
  task1 = "Calculate mean of iris$Sepal.Length",
  task2 = "Create a boxplot of iris data",
  task3 = "Run t-test on iris species"
)
```

```
results <- Execute_Tasks(
  tasks = tasks,
  llm_client = my_llm_provider,
  execution_engine = "tidyprompt"
)

# === llmflow mode ===
results <- Execute_Tasks(
  tasks = tasks,
  llm_client = my_chat_obj,
  execution_engine = "llmflow",
  pkgs_to_use = c("dplyr", "ggplot2"),
  persist_session = TRUE # Reuse session across tasks
)

## End(Not run)
```

Index