

Package: mLLMCelltype (via r-universe)

July 21, 2026

Type Package

Title Cell Type Annotation Using Large Language Models

Version 2.0.7

Author Chen Yang [aut, cre, cph]

Maintainer Chen Yang <cafferychen777@tamu.edu>

Description Automated cell type annotation for single-cell RNA sequencing data using consensus predictions from multiple large language models. Integrates with Seurat objects and provides uncertainty quantification for annotations. Supports various LLM providers including OpenAI, Anthropic, and Google. For details see Yang et al. (2026)
<[doi:10.1038/s42003-026-10420-8](https://doi.org/10.1038/s42003-026-10420-8)>.

License MIT + file LICENSE

BugReports <https://github.com/cafferychen777/mLLMCelltype/issues>

URL <https://cafferyyang.com/mLLMCelltype/>

Encoding UTF-8

Imports httr (>= 1.4.0), jsonlite (>= 1.7.0), R6 (>= 2.5.0), digest (>= 0.6.25), stats, tools, utils

Suggests knitr, rmarkdown, Seurat, testthat

RoxygenNote 7.3.3

Config/build/clean-inst-doc TRUE

VignetteBuilder knitr

NeedsCompilation no

Repository <https://cran.r-universe.dev>

Date/Publication 2026-07-21 08:20:09 UTC

RemoteUrl <https://github.com/cran/mLLMCelltype>

RemoteRef HEAD

RemoteSha cd4b05eb09367d71bf3a023b6ca30a6a6e019432

Contents

annotate_cell_types	2
AnthropicProcessor	7
BaseAPIProcessor	8
CacheManager	10
compare_model_predictions	13
configure_logger	15
create_annotation_prompt	15
create_reasoning_annotation_prompt	16
DeepSeekProcessor	16
GeminiProcessor	18
get_api_key	19
get_logger	20
get_provider	20
GrokProcessor	21
interactive_consensus_annotation	22
KimiProcessor	24
list_custom_models	26
list_custom_providers	26
logging_functions	26
MinimaxProcessor	27
mllmcelltype_cache_dir	28
mllmcelltype_clear_cache	29
OpenAIProcessor	29
OpenRouterProcessor	31
QwenProcessor	32
register_custom_model	34
register_custom_provider	35
StepFunProcessor	36
UnifiedLogger	37
ZhipuProcessor	42
Index	44

annotate_cell_types	<i>Cell Type Annotation with Multi-LLM Framework</i>
---------------------	--

Description

A comprehensive function for automated cell type annotation using multiple Large Language Models (LLMs). This function supports both Seurat's differential gene expression results and custom gene lists as input. It implements a sophisticated annotation pipeline that leverages state-of-the-art LLMs to identify cell types based on marker gene expression patterns.

- A data frame from Seurat's FindAllMarkers() function containing differential gene expression results (must have columns: 'cluster', 'gene', and 'avg_log2FC'). The function will select the top genes based on avg_log2FC for each cluster.

- A list where each element has a 'genes' field containing marker genes for a cluster. This can be in one of these formats:
 - Named with cluster IDs: `list("0" = list(genes = c(...)), "1" = list(genes = c(...)))`
 - Named with cell type names: `list(t_cells = list(genes = c(...)), b_cells = list(genes = c(...)))`
 - Unnamed list: `list(list(genes = c(...)), list(genes = c(...)))`
- Cluster IDs are preserved as-is. The function does not modify or re-index cluster IDs. ('mouse brain'). This helps provide context for more accurate annotations.
- OpenAI: 'gpt-5.5', 'gpt-5.4', 'gpt-5.4-mini'
- Anthropic: 'claude-opus-4-7', 'claude-opus-4-6', 'claude-sonnet-4-6', 'claude-haiku-4-5-20251001'
- DeepSeek: 'deepseek-v4-flash', 'deepseek-v4-pro'
- Google: 'gemini-3.1-pro-preview', 'gemini-3-flash-preview', 'gemini-3.1-flash-lite'
- Alibaba: 'qwen3.6-max-preview', 'qwen3.6-plus', 'qwen3.6-flash'
- Stepfun: 'step-3.5-flash', 'step-3.5-flash-2603', 'step-3'
- Zhipu/Z.AI: 'glm-5.1', 'glm-5-turbo', 'glm-5'
- MiniMax: 'MiniMax-M2.7', 'MiniMax-M2.7-highspeed', 'MiniMax-M2.5'
- X.AI: 'grok-4.3', 'grok-4.3-latest', 'grok-latest'
- OpenRouter: Provides access to models from multiple providers through a single API. Format: 'provider/model-name'
 - OpenAI models: 'openai/gpt-5.5', 'openai/gpt-5.4-mini'
 - Anthropic models: 'anthropic/claude-opus-4.7', 'anthropic/claude-sonnet-4.6'
 - Google models: 'google/gemini-3.1-pro-preview', 'google/gemini-3-flash-preview'
 - X.AI models: 'x-ai/grok-4.3'
 - Stepfun models: 'stepfun/step-3.5-flash' Each provider requires a specific API key format and authentication method:
- OpenAI: "sk-..." (obtain from OpenAI platform)
- Anthropic: "sk-ant-..." (obtain from Anthropic console)
- Google: A Google API key for Gemini models (obtain from Google AI)
- DeepSeek: API key from DeepSeek platform
- Qwen: API key from Alibaba Cloud
- Stepfun: API key from Stepfun AI
- Zhipu: API key from Zhipu AI
- MiniMax: API key from MiniMax
- X.AI: API key for Grok models
- OpenRouter: "sk-or-..." (obtain from OpenRouter) OpenRouter provides access to multiple models through a single API key

The API key can be provided directly or stored in environment variables:

```
# Direct API key
result <- annotate_cell_types(input, tissue_name, model="gpt-5.5",
                             api_key="sk-...")

# Using environment variables
Sys.setenv(OPENAI_API_KEY="sk-...")
Sys.setenv(ANTHROPIC_API_KEY="sk-ant-...")
Sys.setenv(OPENROUTER_API_KEY="sk-or-...")

# Then use with environment variables
result <- annotate_cell_types(input, tissue_name, model="claude-sonnet-4-6",
                             api_key=Sys.getenv("ANTHROPIC_API_KEY"))
```

If NA, returns the generated prompt without making an API call, which is useful for reviewing the prompt before sending it to the API. when input is from Seurat's FindAllMarkers(). Default: 10

- A single character string: Applied to all providers (e.g., "https://api.proxy.com/v1")
- A named list: Provider-specific URLs (e.g., list(openai = "https://openai-proxy.com/v1", anthropic = "https://anthropic-proxy.com/v1")). This is useful for:
 - Users accessing international APIs through proxies
 - Enterprise users with internal API gateways
 - Development/testing with local or alternative endpoints If NULL (default), uses official API endpoints for each provider.

Usage

```
annotate_cell_types(
  input,
  tissue_name,
  model = "gpt-5.5",
  api_key = NA,
  top_gene_count = 10,
  debug = FALSE,
  base_urls = NULL,
  return_reasoning = FALSE
)
```

Arguments

input	Either a data frame from Seurat's FindAllMarkers() containing columns 'cluster', 'gene', and 'avg_log2FC', or a list with 'genes' field for each cluster
tissue_name	Required tissue context (e.g., 'human PBMC', 'mouse brain') for more accurate annotations
model	Model name to use. Default: 'gpt-5.5'. See details for supported models
api_key	API key for the selected model provider as a non-empty character scalar. If NA, returns prompt only.
top_gene_count	Number of top genes to use per cluster when input is from Seurat. Default: 10

debug	Logical indicating whether to enable debug output. Default: FALSE
base_urls	Optional base URLs for API endpoints. Can be a string or named list for custom endpoints
return_reasoning	Logical. If TRUE, returns a structured list per cluster containing cell_type, marker_genes, and gene_expression fields instead of plain labels. Default: FALSE.

Value

When `api_key` is provided and `return_reasoning` is FALSE, the provider response split by newline as a character vector. When `return_reasoning` is TRUE, a named list where each element is a list with `cell_type`, `marker_genes`, and `gene_expression`. When `api_key` is NA, the generated prompt string.

See Also

- [Seurat::FindAllMarkers\(\)](#)
- [get_provider\(\)](#)

Examples

```
# Example 1: Using custom gene lists, returning prompt only (no API call)
annotate_cell_types(
  input = list(
    t_cells = list(genes = c('CD3D', 'CD3E', 'CD3G', 'CD28')),
    b_cells = list(genes = c('CD19', 'CD79A', 'CD79B', 'MS4A1')),
    monocytes = list(genes = c('CD14', 'CD68', 'CSF1R', 'FCGR3A'))
  ),
  tissue_name = 'human PBMC',
  model = 'gpt-5.5',
  api_key = NA # Returns prompt only without making API call
)

# Example 2: Using with Seurat pipeline and OpenAI model
## Not run:
library(Seurat)

# Load example data
data("pbmc_small")

# Find marker genes
all.markers <- FindAllMarkers(
  object = pbmc_small,
  only.pos = TRUE,
  min.pct = 0.25,
  logfc.threshold = 0.25
)

# Set API key in environment variable (recommended approach)
Sys.setenv(OPENAI_API_KEY = "your-openai-api-key")
```

```

# Get cell type annotations using OpenAI model
openai_annotations <- annotate_cell_types(
  input = all.markers,
  tissue_name = 'human PBMC',
  model = 'gpt-5.5',
  api_key = Sys.getenv("OPENAI_API_KEY"),
  top_gene_count = 15
)

# Example 3: Using Anthropic Claude model
Sys.setenv(ANTHROPIC_API_KEY = "your-anthropic-api-key")

claude_annotations <- annotate_cell_types(
  input = all.markers,
  tissue_name = 'human PBMC',
  model = 'claude-opus-4-7',
  api_key = Sys.getenv("ANTHROPIC_API_KEY"),
  top_gene_count = 15
)

# Example 4: Using OpenRouter to access multiple models
Sys.setenv(OPENROUTER_API_KEY = "your-openrouter-api-key")

# Access OpenAI models through OpenRouter
openrouter_gpt4_annotations <- annotate_cell_types(
  input = all.markers,
  tissue_name = 'human PBMC',
  model = 'openai/gpt-5.5', # Note the provider/model format
  api_key = Sys.getenv("OPENROUTER_API_KEY"),
  top_gene_count = 15
)

# Access Anthropic models through OpenRouter
openrouter_claude_annotations <- annotate_cell_types(
  input = all.markers,
  tissue_name = 'human PBMC',
  model = 'anthropic/claude-opus-4.6', # Note the provider/model format
  api_key = Sys.getenv("OPENROUTER_API_KEY"),
  top_gene_count = 15
)

# Example 5: Using with mouse brain data
mouse_annotations <- annotate_cell_types(
  input = mouse_markers, # Your mouse marker genes
  tissue_name = 'mouse brain', # Specify correct tissue for context
  model = 'gpt-5.5',
  api_key = Sys.getenv("OPENAI_API_KEY"),
  top_gene_count = 20, # Use more genes for complex tissues
  debug = TRUE # Enable debug output
)

## End(Not run)

```

AnthropicProcessor	<i>Anthropic API Processor</i>
--------------------	--------------------------------

Description

Anthropic API Processor

Anthropic API Processor

Details

Concrete implementation of BaseAPIProcessor for Anthropic models. Handles Anthropic-specific API calls, authentication, and response parsing.

Super class

`mLLMCelltype::BaseAPIProcessor` -> AnthropicProcessor

Methods

Public methods:

- `AnthropicProcessor$new()`
- `AnthropicProcessor$get_default_api_url()`
- `AnthropicProcessor$make_api_call()`
- `AnthropicProcessor$extract_response_content()`
- `AnthropicProcessor$extract_usage()`
- `AnthropicProcessor$clone()`

Method `new()`: Initialize Anthropic processor

Usage:

`AnthropicProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default Anthropic API URL

Usage:

`AnthropicProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to Anthropic

Usage:

`AnthropicProcessor$make_api_call(chunk_content, model, api_key)`

Arguments:

`chunk_content` Prompt text to send

model Model identifier
 api_key Anthropic API key

Method `extract_response_content()`: Extract response content from Anthropic API response

Usage:

`AnthropicProcessor$extract_response_content(response, model)`

Arguments:

response HTTP response object
 model Model identifier

Method `extract_usage()`: Extract normalized Anthropic token usage

Usage:

`AnthropicProcessor$extract_usage(response)`

Arguments:

response HTTP response object

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

Usage:

`AnthropicProcessor$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

BaseAPIProcessor	<i>Base API Processor Class</i>
------------------	---------------------------------

Description

Base API Processor Class

Base API Processor Class

Details

Abstract base class for API processors that provides common functionality including unified logging, error handling, input processing, and response validation. This eliminates code duplication across all provider-specific processors.

Public fields

provider_name Name of the API provider
 logger Unified logger instance
 base_url Custom base URL for API endpoints

Methods

Public methods:

- `BaseAPIProcessor$new()`
- `BaseAPIProcessor$process_request()`
- `BaseAPIProcessor$get_api_url()`
- `BaseAPIProcessor$get_default_api_url()`
- `BaseAPIProcessor$make_api_call()`
- `BaseAPIProcessor$extract_response_content()`
- `BaseAPIProcessor$extract_usage()`
- `BaseAPIProcessor$clone()`

Method `new()`: Initialize the base API processor

Usage:

```
BaseAPIProcessor$new(provider_name, base_url = NULL)
```

Arguments:

`provider_name` Provider identifier used for logging and dispatch

`base_url` Optional custom API endpoint

Method `process_request()`: Main entry point for processing API requests

Usage:

```
BaseAPIProcessor$process_request(prompt, model, api_key, normalize = TRUE)
```

Arguments:

`prompt` Prompt text to send

`model` Model identifier

`api_key` Provider API key

`normalize` Logical. If TRUE (default), the response is normalized into non-empty trimmed lines. If FALSE, the raw response string is returned instead. Set to FALSE when the caller needs the original text (e.g., JSON parsing in reasoning mode).

Method `get_api_url()`: Get the API URL to use for requests

Usage:

```
BaseAPIProcessor$get_api_url()
```

Method `get_default_api_url()`: Abstract method to be implemented by subclasses for getting default API URL

Usage:

```
BaseAPIProcessor$get_default_api_url()
```

Method `make_api_call()`: Abstract method to be implemented by subclasses for making the actual API call

Usage:

```
BaseAPIProcessor$make_api_call(chunk_content, model, api_key)
```

Arguments:

chunk_content Prompt text to send
 model Model identifier
 api_key Provider API key

Method `extract_response_content()`: Abstract method to be implemented by subclasses for extracting content from response

Usage:

`BaseAPIProcessor$extract_response_content(response, model)`

Arguments:

response HTTP response object
 model Model identifier

Method `extract_usage()`: Extract normalized token usage from a provider response

Usage:

`BaseAPIProcessor$extract_usage(response)`

Arguments:

response HTTP response object Make API call and extract response content

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

Usage:

`BaseAPIProcessor$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

CacheManager

Cache Manager Class

Description

Manages caching of consensus analysis results

Public fields

cache_dir Directory to store cache files. Options:

- NULL (default): Uses system cache directory
- "local": Uses .mllmcelltype_cache in current directory
- "temp": Uses temporary directory
- Custom path: Any other string is used as directory path

cache_version Current cache version

Methods**Public methods:**

- `CacheManager$new()`
- `CacheManager$get_cache_dir()`
- `CacheManager$generate_key()`
- `CacheManager$save_to_cache()`
- `CacheManager$load_from_cache()`
- `CacheManager$has_cache()`
- `CacheManager$get_cache_stats()`
- `CacheManager$clear_cache()`
- `CacheManager$validate_cache()`
- `CacheManager$clone()`

Method new(): Initialize cache manager

- NULL (default): Uses system cache directory via `tools::R_user_dir()`
- "local": Uses `.mlmcelltype_cache` in current directory
- "temp": Uses temporary directory (cleared on R restart)
- Custom path: Any other string is used as directory path

Usage:

```
CacheManager$new(cache_dir = NULL)
```

Arguments:

`cache_dir` Cache directory selector or custom path

Method get_cache_dir(): Get actual cache directory path*Usage:*

```
CacheManager$get_cache_dir()
```

Method generate_key(): Generate cache key from input parameters (improved version)*Usage:*

```
CacheManager$generate_key(
  input,
  models,
  cluster_id,
  tissue_name = "",
  top_gene_count = 10,
  discussion_context = NULL
)
```

Arguments:

`input` Marker gene input

`models` Model identifiers contributing to the result

`cluster_id` Cluster identifier

`tissue_name` Tissue context included in the cache key

`top_gene_count` Number of marker genes used

discussion_context Optional normalized inputs that affect a discussion

Method save_to_cache(): Save results to cache

Usage:

CacheManager\$save_to_cache(key, data)

Arguments:

key Valid cache key

data Result object to persist

Method load_from_cache(): Load results from cache

Usage:

CacheManager\$load_from_cache(key)

Arguments:

key Valid cache key

Method has_cache(): Check if results exist in cache

Usage:

CacheManager\$has_cache(key)

Arguments:

key Valid cache key

Method get_cache_stats(): Get cache statistics

Usage:

CacheManager\$get_cache_stats()

Method clear_cache(): Clear all cache

Usage:

CacheManager\$clear_cache(confirm = FALSE)

Arguments:

confirm Whether deletion is explicitly confirmed

Method validate_cache(): Validate cache content

Usage:

CacheManager\$validate_cache(key)

Arguments:

key Valid cache key Return deterministic paths for regular RDS cache files Resolve a validated cache key to a path inside the cache directory Write one R object to a path; separated for deterministic failure testing Persist cache data through a same-directory temporary file Create stable hash from genes list Create stable hash from models list Create stable hash from tissue_name and top_gene_count Create stable hash from cluster ID

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

Usage:

CacheManager\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

compare_model_predictions

Compare predictions from different models

Description

This function runs the same input through multiple models and compares their predictions. It provides both individual predictions and a consensus analysis.

Usage

```
compare_model_predictions(
  input,
  tissue_name,
  models = c("claude-opus-4-7", "gpt-5.5", "gemini-3.1-pro-preview", "deepseek-v4-flash",
    "qwen3.6-plus", "grok-4.3"),
  api_keys,
  top_gene_count = 10,
  consensus_threshold = 0.5,
  base_urls = NULL
)
```

Arguments

input	Either a data frame from Seurat's FindAllMarkers() containing columns 'cluster', 'gene', and 'avg_log2FC', or a list with 'genes' field for each cluster
tissue_name	Tissue context (e.g., 'human PBMC', 'mouse brain') for more accurate annotations
models	Vector of model names to use for comparison. Default includes top models from each provider
api_keys	Named list of API keys for the models, with provider or model names as keys. Every model in models must resolve to a non-NULL API key.
top_gene_count	Number of top genes to use per cluster when input is from Seurat. Default: 10
consensus_threshold	Minimum agreement threshold for consensus (0-1). Default: 0.5. Consensus is only evaluated when at least two non-missing model predictions are available for a cluster.
base_urls	Optional base URLs for API endpoints. Can be a string or named list for provider-specific custom endpoints.

Value

List containing individual model predictions and consensus analysis. If a cluster has fewer than two valid predictions after alignment/padding, its consensus-related outputs are NA.

Note

This function uses `create_standardization_prompt` from `prompt_templates.R` Supported models:

- OpenAI: 'gpt-5.5', 'gpt-5.4', 'gpt-5.4-mini'
 - Anthropic: 'claude-opus-4-7', 'claude-opus-4-6', 'claude-sonnet-4-6', 'claude-haiku-4-5-20251001'
 - DeepSeek: 'deepseek-v4-flash', 'deepseek-v4-pro'
 - Google: 'gemini-3.1-pro-preview', 'gemini-3-flash-preview', 'gemini-3.1-flash-lite'
 - Alibaba: 'qwen3.6-max-preview', 'qwen3.6-plus', 'qwen3.6-flash'
 - Stepfun: 'step-3.5-flash', 'step-3.5-flash-2603', 'step-3'
 - Zhipu/Z.AI: 'glm-5.1', 'glm-5-turbo', 'glm-5'
 - MiniMax: 'MiniMax-M2.7', 'MiniMax-M2.7-highspeed', 'MiniMax-M2.5'
 - X.AI: 'grok-4.3', 'grok-4.3-latest', 'grok-latest'
 - OpenRouter: Provides access to models from multiple providers through a single API. Format: 'provider/model-name'
 - OpenAI models: 'openai/gpt-5.5', 'openai/gpt-5.4-mini'
 - Anthropic models: 'anthropic/claude-opus-4.7', 'anthropic/claude-sonnet-4.6'
 - Google models: 'google/gemini-3.1-pro-preview', 'google/gemini-3-flash-preview'
 - X.AI models: 'x-ai/grok-4.3'
 - Stepfun models: 'stepfun/step-3.5-flash'
1. With provider names as keys: `list("openai" = "sk-...", "anthropic" = "sk-ant-...", "openrouter" = "sk-or-...")`
 2. With model names as keys: `list("gpt-5.5" = "sk-...", "claude-sonnet-4-6" = "sk-ant-...")`

The system first tries to find the API key using the provider name. If not found, it then tries using the model name. Example:

```
api_keys <- list(
  "openai" = Sys.getenv("OPENAI_API_KEY"),
  "anthropic" = Sys.getenv("ANTHROPIC_API_KEY"),
  "openrouter" = Sys.getenv("OPENROUTER_API_KEY"),
  "claude-opus-4-7" = "your-claude-opus-key"
)
```

Examples

```
## Not run:
# Compare predictions using different models
api_keys <- list(
  "claude-sonnet-4-6" = "your-anthropic-key",
  "deepseek-v4-pro" = "your-deepseek-key",
  "gemini-3.1-pro-preview" = "your-gemini-key",
  "qwen3.6-plus" = "your-qwen-key"
)

results <- compare_model_predictions(
```

```
input = list(gs1=c('CD4','CD3D'), gs2='CD14'),
tissue_name = 'PBMC',
api_keys = api_keys
)

## End(Not run)
```

configure_logger	<i>Set global logger configuration</i>
------------------	--

Description

Set global logger configuration

Usage

```
configure_logger(level = "INFO", console_output = FALSE, json_format = TRUE)
```

Arguments

- level Logging level: "DEBUG", "INFO", "WARN", or "ERROR". Default: "INFO"
- console_output Whether to enable console output. Default: FALSE
- json_format Whether to use JSON format for log messages. Default: TRUE

Value

Invisible logger object

create_annotation_prompt	<i>Create prompt for cell type annotation</i>
--------------------------	---

Description

Create prompt for cell type annotation

Usage

```
create_annotation_prompt(input, tissue_name, top_gene_count = 10)
```

Arguments

- input Either a data frame from Seurat's FindAllMarkers() or a list for each cluster where each element is either a character vector of genes or a list containing a genes field Cluster IDs in named inputs are preserved as-is; unnamed list input receives sequential IDs starting at "0".
- tissue_name Tissue context for the annotation (e.g., 'human PBMC', 'mouse brain')
- top_gene_count Number of top genes to use per cluster when input is from Seurat. Default: 10

Value

A list with prompt (formatted prompt text), expected_count (number of clusters), and gene_lists (cluster ID to marker genes mapping).

create_reasoning_annotation_prompt	Create reasoning-aware prompt for cell type annotation
------------------------------------	--

Description

Create reasoning-aware prompt for cell type annotation

Usage

```
create_reasoning_annotation_prompt(input, tissue_name, top_gene_count = 10)
```

Arguments

input	Either a data frame from Seurat’s FindAllMarkers() or a list for each cluster where each element is either a character vector of genes or a list containing a genes field.
tissue_name	Tissue context for the annotation (e.g., ‘human PBMC’, ‘mouse brain’)
top_gene_count	Number of top genes to use per cluster when input is from Seurat. Default: 10

Value

A list with prompt (formatted prompt text), expected_count (number of clusters), and gene_lists (cluster ID to marker genes mapping).

DeepSeekProcessor	DeepSeek API Processor
-------------------	------------------------

Description

DeepSeek API Processor
DeepSeek API Processor

Details

Concrete implementation of BaseAPIProcessor for DeepSeek models. Handles DeepSeek-specific API calls, authentication, and response parsing.

Super class

mLLMCelltype::BaseAPIProcessor -> DeepSeekProcessor

Methods

Public methods:

- `DeepSeekProcessor$new()`
- `DeepSeekProcessor$get_default_api_url()`
- `DeepSeekProcessor$make_api_call()`
- `DeepSeekProcessor$extract_response_content()`
- `DeepSeekProcessor$clone()`

Method `new()`: Initialize DeepSeek processor

Usage:

`DeepSeekProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default DeepSeek API URL

Usage:

`DeepSeekProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to DeepSeek

Usage:

`DeepSeekProcessor$make_api_call(chunk_content, model, api_key)`

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier

`api_key` DeepSeek API key

Method `extract_response_content()`: Extract response content from DeepSeek API response

Usage:

`DeepSeekProcessor$extract_response_content(response, model)`

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

`DeepSeekProcessor$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

GeminiProcessor

Gemini API Processor

Description

Gemini API Processor

Gemini API Processor

Details

Concrete implementation of BaseAPIProcessor for Gemini models. Handles Gemini-specific API calls, authentication, and response parsing.

Super class

`mLLMCelltype::BaseAPIProcessor` -> GeminiProcessor

Methods

Public methods:

- `GeminiProcessor$new()`
- `GeminiProcessor$get_default_api_url()`
- `GeminiProcessor$get_api_url_for_model()`
- `GeminiProcessor$make_api_call()`
- `GeminiProcessor$extract_response_content()`
- `GeminiProcessor$extract_usage()`
- `GeminiProcessor$clone()`

Method `new()`: Initialize Gemini processor

Usage:

`GeminiProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default Gemini API URL template

Usage:

`GeminiProcessor$get_default_api_url()`

Method `get_api_url_for_model()`: Get API URL for specific model

Usage:

`GeminiProcessor$get_api_url_for_model(model)`

Arguments:

`model` Model identifier

Method make_api_call(): Make API call to Gemini

Usage:

```
GeminiProcessor$make_api_call(chunk_content, model, api_key)
```

Arguments:

chunk_content Prompt text to send

model Model identifier

api_key Gemini API key

Method extract_response_content(): Extract response content from Gemini API response

Usage:

```
GeminiProcessor$extract_response_content(response, model)
```

Arguments:

response HTTP response object

model Model identifier

Method extract_usage(): Extract normalized Gemini token usage

Usage:

```
GeminiProcessor$extract_usage(response)
```

Arguments:

response HTTP response object

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

get_api_key	Get an API key for a model
-------------	----------------------------

Description

Retrieves a configured API key by checking the model’s provider name first, followed by the exact model name.

Usage

```
get_api_key(model, api_keys)
```

Arguments

model	Model name to get API key for
api_keys	Named list of API keys with provider or model names as keys

Value

A trimmed API key string, or NULL when no valid key is configured.

get_logger	Get the global logger instance
------------	--------------------------------

Description

Get the global logger instance

Usage

get_logger()

get_provider	Determine provider from model name
--------------	------------------------------------

Description

This function determines the appropriate provider (e.g., OpenAI, Anthropic, Google, OpenRouter) based on the model name. Uses prefix-based matching for efficient and maintainable provider detection. New models following existing naming conventions are automatically supported.

Usage

get_provider(model)

Arguments

model Character string specifying the model name (e.g., "gpt-5.5", "claude-opus-4-7").

Details

Supported providers and model prefixes:

- OpenAI: gpt-, o1, o3*, o4*, chatgpt-, codex- (e.g., 'gpt-5.5', 'gpt-5.4-mini')
- Anthropic: claude-* (e.g., 'claude-opus-4-7', 'claude-sonnet-4-6')
- DeepSeek: deepseek-* (e.g., 'deepseek-v4-flash', 'deepseek-v4-pro')
- Google: gemini-* (e.g., 'gemini-3.1-pro-preview', 'gemini-3-flash-preview')
- Qwen: qwen*, qwq-* (e.g., 'qwen3.6-plus', 'qwen3.6-flash')
- Stepfun: step-* (e.g., 'step-3.5-flash', 'step-3')
- Zhipu: glm-, chatglm (e.g., 'glm-5.1', 'glm-5-turbo')
- MiniMax: minimax-* (e.g., 'MiniMax-M2.7', 'MiniMax-M2.5')
- Grok: grok-* (e.g., 'grok-4.3', 'grok-4.3-latest')
- Kimi (Moonshot AI Open Platform): kimi-, moonshot- (e.g., 'kimi-k2.6', 'moonshot-v1-8k')
- OpenRouter: Any model with '/' in the name (e.g., 'openai/gpt-5.5', 'anthropic/claude-opus-4.7')

Value

Character string of the provider name (e.g., "openai", "anthropic").

GrokProcessor	<i>Grok API Processor</i>
---------------	---------------------------

Description

Grok API Processor
Grok API Processor

Details

Concrete implementation of BaseAPIProcessor for Grok models. Handles Grok-specific API calls, authentication, and response parsing.

Super class

mLLMCelltype: BaseAPIProcessor -> GrokProcessor

Methods

Public methods:

- GrokProcessor\$new()
- GrokProcessor\$get_default_api_url()
- GrokProcessor\$make_api_call()
- GrokProcessor\$extract_response_content()
- GrokProcessor\$clone()

Method new(): Initialize Grok processor

Usage:

GrokProcessor\$new(base_url = NULL)

Arguments:

base_url Optional custom API endpoint

Method get_default_api_url(): Get default Grok API URL

Usage:

GrokProcessor\$get_default_api_url()

Method make_api_call(): Make API call to Grok

Usage:

GrokProcessor\$make_api_call(chunk_content, model, api_key)

Arguments:

chunk_content Prompt text to send

model Model identifier
api_key Grok API key

Method `extract_response_content()`: Extract response content from Grok API response

Usage:

`GrokProcessor$extract_response_content(response, model)`

Arguments:

response HTTP response object
model Model identifier

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

Usage:

`GrokProcessor$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

interactive_consensus_annotation

Interactive consensus building for cell type annotation

Description

This function implements an interactive voting and discussion mechanism where multiple LLMs collaborate to reach a consensus on cell type annotations, particularly focusing on clusters with low agreement. The process includes:

1. Initial voting by all LLMs
2. Identification of controversial clusters
3. Detailed discussion for controversial clusters
4. Final summary by a designated LLM (default: Claude)

Usage

```
interactive_consensus_annotation(
  input,
  tissue_name,
  models = c("claude-opus-4-7", "gpt-5.5", "gemini-3.1-pro-preview", "deepseek-v4-flash",
    "grok-4.3"),
  api_keys,
  top_gene_count = 10,
  controversy_threshold = 0.7,
  entropy_threshold = 1,
  max_discussion_rounds = 3,
  consensus_check_model = NULL,
```

```

    log_dir = "logs",
    cache_dir = NULL,
    use_cache = TRUE,
    base_urls = NULL,
    clusters_to_analyze = NULL,
    force_rerun = FALSE
)

```

Arguments

input	Either a data frame from Seurat's FindAllMarkers() function containing differential gene expression results (must have columns: 'cluster', 'gene', and 'avg_log2FC'), or a list where each element is either a character vector of genes or a list containing a genes field.
tissue_name	Character string specifying the tissue type for context-aware cell type annotation (e.g., 'human PBMC', 'mouse brain'). Required.
models	Character vector of model names to use for consensus annotation. Minimum 2 models required. Supports models from OpenAI, Anthropic, DeepSeek, Google, Alibaba, Stepfun, Zhipu, MiniMax, X.AI, and OpenRouter.
api_keys	Named, non-empty list of API keys. Can use provider names as keys (e.g., "openai", "anthropic") or model names as keys (e.g., "gpt-5").
top_gene_count	Integer specifying the number of top marker genes to use for annotation per cluster (default: 10).
controversy_threshold	Numeric value between 0 and 1 for consensus proportion threshold. Clusters below this threshold are considered controversial (default: 0.7).
entropy_threshold	Numeric value for entropy threshold. Higher entropy indicates more disagreement among models (default: 1.0).
max_discussion_rounds	Integer specifying maximum number of discussion rounds for controversial clusters (default: 3).
consensus_check_model	Character string specifying which model to use for consensus checking. If NULL, uses the first model that succeeds during initial annotation.
log_dir	Character scalar specifying directory for log files (default: "logs"). This function reinitializes the session logger with this directory at the start of each call.
cache_dir	Character string or NULL. Cache directory for storing results. NULL uses system cache, "local" uses current directory, "temp" uses temporary directory, or specify custom path.
use_cache	Logical indicating whether to use caching (default: TRUE).
base_urls	Named list or character string specifying custom API base URLs. Useful for proxies or alternative endpoints. If NULL, uses official endpoints.
clusters_to_analyze	Character or numeric vector specifying which clusters to analyze. If NULL (default), all clusters are analyzed.

force_rerun Logical indicating whether to force rerun of all specified clusters, ignoring cache. Only affects controversial cluster discussions (default: FALSE).

Value

- A list containing:
- initial_results: Initial voting results, consensus checks, and controversial cluster IDs
 - final_annotations: Final annotations keyed by cluster ID
 - controversial_clusters: Clusters identified as controversial
 - discussion_logs: Detailed discussion logs for controversial clusters
 - session_id: Logger session identifier
 - voting_results: Backward-compatible alias of initial_results
 - discussion_results: Backward-compatible alias of discussion_logs
 - final_consensus: Backward-compatible alias of final_annotations

KimiProcessor	<i>Kimi API Processor</i>
---------------	---------------------------

Description

Kimi API Processor
Kimi API Processor

Details

Concrete implementation of BaseAPIProcessor for Kimi models. By default it targets the Moonshot AI Open Platform over the OpenAI-compatible Chat Completions protocol, with k2 thinking mode disabled for deterministic output. A custom base_url may instead point at the Kimi Code platform (api.kimi.com/coding), which speaks both protocols; the protocol is inferred from the effective endpoint URL. URLs ending in '/messages' use the Anthropic Messages protocol; the Kimi Code base 'https://api.kimi.com/coding' and URLs ending in '/chat/completions' use OpenAI-compatible Chat Completions.

Super class

mLLMCelltype::BaseAPIProcessor -> KimiProcessor

Methods

- Public methods:
- KimiProcessor\$new()
 - KimiProcessor\$get_default_api_url()
 - KimiProcessor\$make_api_call()
 - KimiProcessor\$extract_response_content()

- `KimiProcessor$extract_usage()`
- `KimiProcessor$clone()`

Method `new()`: Initialize Kimi processor

Usage:

```
KimiProcessor$new(base_url = NULL)
```

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default Kimi API URL

Usage:

```
KimiProcessor$get_default_api_url()
```

Method `make_api_call()`: Make API call to Kimi

Usage:

```
KimiProcessor$make_api_call(chunk_content, model, api_key)
```

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier (e.g., 'kimi-k2.6', 'moonshot-v1-8k')

`api_key` Moonshot API key

Method `extract_response_content()`: Extract response content from Kimi API response

Usage:

```
KimiProcessor$extract_response_content(response, model)
```

Arguments:

`response` HTTP response object

`model` Model identifier

Method `extract_usage()`: Extract normalized token usage from a Kimi API response

Usage:

```
KimiProcessor$extract_usage(response)
```

Arguments:

`response` HTTP response object

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

list_custom_models	<i>Get list of registered custom models</i>
--------------------	---

Description

Get list of registered custom models

Usage

```
list_custom_models()
```

list_custom_providers	<i>Get list of registered custom providers</i>
-----------------------	--

Description

Get list of registered custom providers

Usage

```
list_custom_providers()
```

logging_functions	<i>Convenience functions for logging</i>
-------------------	--

Description

Convenience functions for logging

Usage

```
log_debug(message, context = NULL)
```

```
log_info(message, context = NULL)
```

```
log_warn(message, context = NULL)
```

```
log_error(message, context = NULL)
```

Arguments

message	Log message string
context	Optional context information (list or character)

Value

Invisible NULL

MinimaxProcessor	<i>Minimax API Processor</i>
------------------	------------------------------

Description

Minimax API Processor

Minimax API Processor

Details

Concrete implementation of BaseAPIProcessor for Minimax models. Handles Minimax-specific API calls, authentication, and response parsing.

Super class

`mLLMCelltype::BaseAPIProcessor` -> MinimaxProcessor

Methods

Public methods:

- `MinimaxProcessor$new()`
- `MinimaxProcessor$get_default_api_url()`
- `MinimaxProcessor$make_api_call()`
- `MinimaxProcessor$extract_response_content()`
- `MinimaxProcessor$clone()`

Method `new()`: Initialize Minimax processor

Usage:

`MinimaxProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default MiniMax OpenAI-compatible chat completions API URL

Usage:

`MinimaxProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to Minimax

Usage:

`MinimaxProcessor$make_api_call(chunk_content, model, api_key)`

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier

`api_key` MiniMax API key

Method `extract_response_content()`: Extract response content from Minimax API response

Usage:

```
MinimaxProcessor$extract_response_content(response, model)
```

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`mllmcelltype_cache_dir`

Get mLLMCelltype cache location

Description

Display the cache directory location

Usage

```
mllmcelltype_cache_dir(cache_dir = NULL)
```

Arguments

`cache_dir` Cache directory specification. NULL uses system default, "local" uses current dir, "temp" uses temp dir, or custom path

Value

Invisible cache directory path

Examples

```
## Not run:
mllmcelltype_cache_dir()
mllmcelltype_cache_dir("local")

## End(Not run)
```

mllmcelltype_clear_cache	<i>Clear mLLMCelltype cache</i>
--------------------------	---------------------------------

Description

Clear the mLLMCelltype cache

Usage

```
mllmcelltype_clear_cache(cache_dir = NULL)
```

Arguments

cache_dir	Cache directory specification. NULL uses system default, "local" uses current dir, "temp" uses temp dir, or custom path
-----------	---

Value

Invisible NULL

Examples

```
## Not run:  
mllmcelltype_clear_cache()  
mllmcelltype_clear_cache("local")  
  
## End(Not run)
```

OpenAIProcessor	<i>OpenAI API Processor</i>
-----------------	-----------------------------

Description

OpenAI API Processor
OpenAI API Processor

Details

Concrete implementation of BaseAPIProcessor for OpenAI models. Handles OpenAI-specific API calls, authentication, and response parsing.

Super class

```
mLLMCelltype::BaseAPIProcessor -> OpenAIProcessor
```

Methods

Public methods:

- `OpenAIProcessor$new()`
- `OpenAIProcessor$get_default_api_url()`
- `OpenAIProcessor$make_api_call()`
- `OpenAIProcessor$extract_response_content()`
- `OpenAIProcessor$clone()`

Method `new()`: Initialize OpenAI processor

Usage:

`OpenAIProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default OpenAI API URL

Usage:

`OpenAIProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to OpenAI

Usage:

`OpenAIProcessor$make_api_call(chunk_content, model, api_key)`

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier

`api_key` OpenAI API key

Method `extract_response_content()`: Extract response content from OpenAI API response

Usage:

`OpenAIProcessor$extract_response_content(response, model)`

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

`OpenAIProcessor$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

OpenRouterProcessor	<i>OpenRouter API Processor</i>
---------------------	---------------------------------

Description

OpenRouter API Processor

OpenRouter API Processor

Details

Concrete implementation of BaseAPIProcessor for OpenRouter models. Handles OpenRouter-specific API calls, authentication, and response parsing.

Super class

`mLLMCelltype::BaseAPIProcessor` -> OpenRouterProcessor

Methods

Public methods:

- `OpenRouterProcessor$new()`
- `OpenRouterProcessor$get_default_api_url()`
- `OpenRouterProcessor$make_api_call()`
- `OpenRouterProcessor$extract_response_content()`
- `OpenRouterProcessor$clone()`

Method `new()`: Initialize OpenRouter processor

Usage:

`OpenRouterProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default OpenRouter API URL

Usage:

`OpenRouterProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to OpenRouter

Usage:

`OpenRouterProcessor$make_api_call(chunk_content, model, api_key)`

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier

`api_key` OpenRouter API key

Method `extract_response_content()`: Extract response content from OpenRouter API response

Usage:

`OpenRouterProcessor$extract_response_content(response, model)`

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

`OpenRouterProcessor$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

QwenProcessor

Qwen API Processor

Description

Qwen API Processor

Qwen API Processor

Details

Concrete implementation of `BaseAPIProcessor` for Qwen models. Handles Qwen-specific API calls, authentication, and response parsing.

Qwen has OpenAI-compatible chat completions endpoints:

- International (US): <https://dashscope-us.aliyuncs.com/compatible-mode/v1/chat/completions>
- Domestic (China): <https://dashscope.aliyuncs.com/compatible-mode/v1/chat/completions>
- Legacy international: <https://dashscope-intl.aliyuncs.com/compatible-mode/v1/chat/completions>
The processor automatically tries international first, then domestic, then legacy international.

Super class

`mLLMCelltype::BaseAPIProcessor` -> QwenProcessor

Methods

Public methods:

- `QwenProcessor$new()`
- `QwenProcessor$get_default_api_url()`
- `QwenProcessor$get_working_api_url()`
- `QwenProcessor$make_api_call()`
- `QwenProcessor$extract_response_content()`
- `QwenProcessor$clone()`

Method `new()`: Test if an endpoint is accessible

Initialize Qwen processor

Usage:

```
QwenProcessor$new(base_url = NULL)
```

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default Qwen OpenAI-compatible chat completions API URL

Usage:

```
QwenProcessor$get_default_api_url()
```

Method `get_working_api_url()`: Get working Qwen API URL with automatic endpoint detection

Usage:

```
QwenProcessor$get_working_api_url(api_key)
```

Arguments:

`api_key` Qwen API key used for regional endpoint probing

Method `make_api_call()`: Make API call to Qwen

Usage:

```
QwenProcessor$make_api_call(chunk_content, model, api_key)
```

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier

`api_key` Qwen API key

Method `extract_response_content()`: Extract response content from Qwen API response

Usage:

```
QwenProcessor$extract_response_content(response, model)
```

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

register_custom_model *Register a custom model for a provider*

Description

Register a custom model for a provider

Usage

```
register_custom_model(model_name, provider_name, model_config = list())
```

Arguments

model_name	Unique name for the custom model
provider_name	Name of the provider this model belongs to
model_config	List of configuration parameters for the model (e.g., temperature, max_tokens)

Value

Invisible TRUE on success

Examples

```
## Not run:
register_custom_model(
  model_name = "my_model",
  provider_name = "my_provider",
  model_config = list(
    temperature = 0.7,
    max_tokens = 2000
  )
)

## End(Not run)
```

`register_custom_provider`*Register a custom LLM provider*

Description

Register a custom LLM provider

Usage

```
register_custom_provider(provider_name, process_fn, description = NULL)
```

Arguments

<code>provider_name</code>	Unique name for the custom provider
<code>process_fn</code>	Function that processes LLM requests. Must accept parameters: <code>prompt</code> , <code>model</code> , <code>api_key</code> ; may optionally accept <code>model_config</code> and <code>base_url</code>
<code>description</code>	Optional description of the provider

Value

Invisible NULL

Examples

```
## Not run:
register_custom_provider(
  provider_name = "my_provider",
  process_fn = function(prompt, model, api_key) {
    # Custom implementation
    response <- httr::POST(
      url = "your_api_endpoint",
      body = list(prompt = prompt),
      encode = "json"
    )
    return(httr::content(response)$choices[[1]]$text)
  }
)

## End(Not run)
```

StepFunProcessor	<i>StepFun API Processor</i>
------------------	------------------------------

Description

StepFun API Processor

StepFun API Processor

Details

Concrete implementation of BaseAPIProcessor for StepFun models. Handles StepFun-specific API calls, authentication, and response parsing.

Super class

`mLLMCelltype::BaseAPIProcessor -> StepFunProcessor`

Methods

Public methods:

- `StepFunProcessor$new()`
- `StepFunProcessor$get_default_api_url()`
- `StepFunProcessor$make_api_call()`
- `StepFunProcessor$extract_response_content()`
- `StepFunProcessor$clone()`

Method `new()`: Initialize StepFun processor

Usage:

`StepFunProcessor$new(base_url = NULL)`

Arguments:

`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default StepFun API URL

Usage:

`StepFunProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to StepFun

Usage:

`StepFunProcessor$make_api_call(chunk_content, model, api_key)`

Arguments:

`chunk_content` Prompt text to send

`model` Model identifier

`api_key` StepFun API key

Method `extract_response_content()`: Extract response content from StepFun API response

Usage:

```
StepFunProcessor$extract_response_content(response, model)
```

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

UnifiedLogger

Unified Logger for mLLMCelltype Package

Description

Unified Logger for mLLMCelltype Package

Unified Logger for mLLMCelltype Package

Details

This logger provides centralized, multi-level logging with structured output, log rotation, and performance monitoring capabilities.

Public fields

`log_dir` Directory for storing log files

`log_level` Current logging level

`session_id` Unique identifier for the current session

`max_log_size` Maximum log file size in MB (default: 10MB)

`max_log_files` Maximum number of log files to keep (default: 5)

`enable_console` Whether to output to console (default: FALSE)

`enable_json` Whether to use JSON format (default: TRUE)

`performance_stats` Performance monitoring statistics

Methods

Public methods:

- `UnifiedLogger$new()`
- `UnifiedLogger$debug()`
- `UnifiedLogger$info()`
- `UnifiedLogger$warn()`
- `UnifiedLogger$error()`
- `UnifiedLogger$log_api_call()`
- `UnifiedLogger$log_api_request_response()`
- `UnifiedLogger$log_cache_operation()`
- `UnifiedLogger$log_cluster_progress()`
- `UnifiedLogger$log_discussion()`
- `UnifiedLogger$log_model_response()`
- `UnifiedLogger$get_performance_summary()`
- `UnifiedLogger$cleanup_logs()`
- `UnifiedLogger$set_level()`
- `UnifiedLogger$clone()`

Method `new()`: Initialize the unified logger

Usage:

```
UnifiedLogger$new(  
  base_dir = "logs",  
  level = "INFO",  
  max_size = 10,  
  max_files = 5,  
  console_output = FALSE,  
  json_format = TRUE  
)
```

Arguments:

`base_dir` Directory for log files
`level` Minimum log level
`max_size` Maximum main log size in megabytes
`max_files` Maximum number of retained main logs
`console_output` Whether to mirror logs to the console
`json_format` Whether main logs use JSON format

Method `debug()`: Log a debug message

Usage:

```
UnifiedLogger$debug(message, context = NULL)
```

Arguments:

`message` Log message
`context` Optional structured context

Method info(): Log an info message

Usage:

```
UnifiedLogger$info(message, context = NULL)
```

Arguments:

message Log message

context Optional structured context

Method warn(): Log a warning message

Usage:

```
UnifiedLogger$warn(message, context = NULL)
```

Arguments:

message Log message

context Optional structured context

Method error(): Log an error message

Usage:

```
UnifiedLogger$error(message, context = NULL)
```

Arguments:

message Log message

context Optional structured context

Method log_api_call(): Log API call performance

Usage:

```
UnifiedLogger$log_api_call(  
  provider,  
  model,  
  duration,  
  success = TRUE,  
  tokens = NULL  
)
```

Arguments:

provider Provider identifier

model Model identifier

duration Request duration in seconds

success Whether the request succeeded

tokens Optional token usage metadata

Method log_api_request_response(): Log complete API request and response for debugging and audit

Usage:

```
UnifiedLogger$log_api_request_response(
  provider,
  model,
  prompt_content,
  response_content,
  request_metadata = NULL,
  response_metadata = NULL
)
```

Arguments:

provider Provider identifier
 model Model identifier
 prompt_content Request prompt
 response_content Provider response or error text
 request_metadata Optional request metadata
 response_metadata Optional response metadata

Method log_cache_operation(): Log cache operations

Usage:

```
UnifiedLogger$log_cache_operation(operation, key, size = NULL)
```

Arguments:

operation Cache operation name
 key Cache key
 size Optional cache object size in bytes

Method log_cluster_progress(): Log cluster annotation progress

Usage:

```
UnifiedLogger$log_cluster_progress(cluster_id, stage, progress = NULL)
```

Arguments:

cluster_id Cluster identifier
 stage Processing stage
 progress Optional progress value

Method log_discussion(): Log detailed cluster discussion with complete model conversations

Usage:

```
UnifiedLogger$log_discussion(cluster_id, event_type, data = NULL)
```

Arguments:

cluster_id Cluster identifier
 event_type Discussion event type
 data Optional event payload

Method log_model_response(): Log model response with concise summary in main log and full text in file

Usage:


```
UnifiedLogger$log_model_response(  
  provider,  
  model,  
  response,  
  stage = "annotation",  
  cluster_id = NULL  
)
```

Arguments:

provider Provider identifier
model Model identifier
response Model response
stage Processing stage
cluster_id Optional cluster identifier

Method `get_performance_summary()`: Get performance summary

Usage:

```
UnifiedLogger$get_performance_summary()
```

Method `cleanup_logs()`: Clean up old log files

Usage:

```
UnifiedLogger$cleanup_logs(force = FALSE)
```

Arguments:

force Whether to remove every main log file

Method `set_level()`: Set logging level

Usage:

```
UnifiedLogger$set_level(level)
```

Arguments:

level Minimum log level

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

ZhipuProcessor	<i>Zhipu API Processor</i>
----------------	----------------------------

Description

Zhipu API Processor
Zhipu API Processor

Details

Concrete implementation of BaseAPIProcessor for Zhipu models. Handles Zhipu-specific API calls, authentication, and response parsing.

Super class

mLLMCelltype::BaseAPIProcessor -> ZhipuProcessor

Methods

Public methods:

- `ZhipuProcessor$new()`
- `ZhipuProcessor$get_default_api_url()`
- `ZhipuProcessor$make_api_call()`
- `ZhipuProcessor$extract_response_content()`
- `ZhipuProcessor$clone()`

Method `new()`: Initialize Zhipu processor

Usage:
`ZhipuProcessor$new(base_url = NULL)`
Arguments:
`base_url` Optional custom API endpoint

Method `get_default_api_url()`: Get default Zhipu API URL

Usage:
`ZhipuProcessor$get_default_api_url()`

Method `make_api_call()`: Make API call to Zhipu

Usage:
`ZhipuProcessor$make_api_call(chunk_content, model, api_key)`
Arguments:
`chunk_content` Prompt text to send
`model` Model identifier
`api_key` Zhipu API key

Method `extract_response_content()`: Extract response content from Zhipu API response

Usage:

```
ZhipuProcessor$extract_response_content(response, model)
```

Arguments:

`response` HTTP response object

`model` Model identifier

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index

annotate_cell_types, [2](#)
AnthropicProcessor, [7](#)

BaseAPIProcessor, [8](#)

CacheManager, [10](#)
compare_model_predictions, [13](#)
configure_logger, [15](#)
create_annotation_prompt, [15](#)
create_reasoning_annotation_prompt, [16](#)

DeepSeekProcessor, [16](#)

GeminiProcessor, [18](#)
get_api_key, [19](#)
get_logger, [20](#)
get_provider, [20](#)
get_provider(), [5](#)
GrokProcessor, [21](#)

interactive_consensus_annotation, [22](#)

KimiProcessor, [24](#)

list_custom_models, [26](#)
list_custom_providers, [26](#)
log_debug (logging_functions), [26](#)
log_error (logging_functions), [26](#)
log_info (logging_functions), [26](#)
log_warn (logging_functions), [26](#)
logging_functions, [26](#)

MinimaxProcessor, [27](#)
mLLMCelltype::BaseAPIProcessor, [7](#), [16](#),
 [18](#), [21](#), [24](#), [27](#), [29](#), [31](#), [32](#), [36](#), [42](#)
mllmcelltype_cache_dir, [28](#)
mllmcelltype_clear_cache, [29](#)

OpenAIProcessor, [29](#)
OpenRouterProcessor, [31](#)

QwenProcessor, [32](#)

register_custom_model, [34](#)
register_custom_provider, [35](#)

Seurat::FindAllMarkers(), [5](#)
StepFunProcessor, [36](#)

UnifiedLogger, [37](#)

ZhipuProcessor, [42](#)