Package: aisdk (via r-universe)

June 2, 2026

Title Unified Interface for AI Model Providers

Version 1.4.12

Description A production-grade AI toolkit for R featuring a layered architecture (Specification, Utilities, Providers, Core), request interception support, robust error handling with exponential retry delays, support for multiple AI model providers ('OpenAI', 'Anthropic', etc.), local small language model inference, distributed 'MCP' ecosystem, multi-agent orchestration, progressive knowledge loading through skills, and a global skill store for sharing AI capabilities.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports R6, httr2, jsonlite, rlang, yaml, callr, processx, memoise, digest, parallel, stats, tools, utils, methods, base64enc, curl

Suggests testthat (>= 3.0.0), httptest2, cli, skimr, evaluate, ggplot2, dplyr, readr, readxl, DBI, RSQLite, rstudioapi, DT, withr, devtools, fs, dotenv, pkgload

Config/testthat/edition 3

URL https://github.com/YuLab-SMU/aisdk, https://yulab-smu.top/aisdk/

BugReports https://github.com/YuLab-SMU/aisdk/issues

NeedsCompilation no

Author Yonghe Xia [aut, cre]

Maintainer Yonghe Xia <xiayh17@gmail.com>

Depends R (>= 4.1.0)

Config/pak/sysreqs libssl-dev

Repository https://yulab-smu.r-universe.dev

Date/Publication 2026-06-02 02:58:18 UTC

RemoteUrl https://github.com/yulab-smu/aisdk

RemoteRef HEAD

RemoteSha bea7986ecd0d5cbf13b479bed094f91f3b177614

Contents

aisdk-package *aisdk: AI SDK for R*

Description

A production-grade AI SDK for R featuring a layered architecture, middleware support, robust error handling, and support for multiple AI model providers.

Architecture

The SDK uses a 4-layer architecture:

- **Specification Layer**: Abstract interfaces (LanguageModelV1, EmbeddingModelV1)
- **Utilities Layer**: Shared tools (HTTP, retry, registry, middleware)
- **Provider Layer**: Concrete implementations (OpenAIProvider, etc.)
- **Core Layer**: High-level API (generate_text, stream_text, embed)

Quick Start

```
library(aisdk)

# Create an OpenAI provider
openai <- create_openai()

# Generate text
result <- generate_text(
  model = openai$language_model("gpt-4o"),
  prompt = "Explain R in one sentence."
)
print(result$text)

# Or use the registry for cleaner syntax
get_default_registry()$register("openai", openai)
result <- generate_text("openai:gpt-4o", "Hello!")
```

Author(s)

Maintainer: Yonghe Xia <xiayh17@gmail.com>

See Also

Useful links:

- https://github.com/YuLab-SMU/aisdk
- https://yulab-smu.top/aisdk/
- Report bugs at https://github.com/YuLab-SMU/aisdk/issues

Agent *Agent Class*

Description

R6 class representing an AI agent. Agents are the worker units in the multi-agent architecture. Each agent has a name, description (for semantic routing), system prompt (persona), and a set of tools it can use.

Key design principle: Agents are stateless regarding conversation history. The ChatSession holds the shared state (history, memory, environment).

Public fields

`name` Unique identifier for this agent.

`description` Description of the agent's capability. This is the "API" that the LLM Manager uses for semantic routing.

`system_prompt` The agent's persona/instructions.

`tools` List of Tool objects this agent can use.

`skill_registry` Optional registry of local skills available to the agent.

`model` Default model ID for this agent.

`capability_models` Optional capability-specific model routes.

Methods

Public methods:

- `Agent$new()`
- `Agent$run()`
- `Agent$stream()`
- `Agent$as_tool()`
- `Agent$create_session()`
- `Agent$print()`
- `Agent$clone()`

Method `new()`**:** Initialize a new Agent.

Usage:

```
Agent$new(
  name,
  description,
  system_prompt = NULL,
  tools = NULL,
  skills = NULL,
  model = NULL,
  capability_models = NULL
)
```

Arguments:

`name` Unique name for this agent (e.g., "DataCleaner", "Visualizer").

`description` A clear description of what this agent does. This is used by the Manager LLM to decide which agent to delegate to.

`system_prompt` Optional system prompt defining the agent's persona.

`tools` Optional list of Tool objects the agent can use.

`skills` Optional character vector of skill paths or "auto" to discover skills. When provided, this automatically loads skills, creates tools, and updates the system prompt.

`model` Optional default model ID for this agent.

`capability_models` Optional named list of capability-specific model routes.

Returns: An Agent object.

Method `run()`**:** Run the agent with a given task.

Usage:

```
Agent$run(
  task,
  session = NULL,
  context = NULL,
  model = NULL,
  max_steps = 10,
  ...
)
```

Arguments:

`task` The task instruction (natural language).

`session` Optional ChatSession for shared state. If NULL, a temporary session is created.

`context` Optional additional context to inject (e.g., from parent agent).

`model` Optional model override. Uses session's model if not provided.

`max_steps` Maximum ReAct loop iterations. Default 10.

`...` Additional arguments passed to generate_text.

Returns: A GenerateResult object from generate_text.

Method `stream()`**:** Run the agent with streaming output.

Usage:

```
Agent$stream(
  task,
  callback = NULL,
  session = NULL,
  context = NULL,
  model = NULL,
  max_steps = 10,
  ...
)
```

Arguments:

`task` The task instruction (natural language).

`callback` Function to handle streaming chunks: callback(text, done).

`session` Optional ChatSession for shared state.

`context` Optional additional context to inject.

`model` Optional model override.

`max_steps` Maximum ReAct loop iterations. Default 10.

`...` Additional arguments passed to stream_text.

Returns: A GenerateResult object (accumulated).

Method `as_tool()`**:** Convert this agent to a Tool.

Usage:

```
Agent$as_tool()
```

Details: This allows the agent to be used as a delegate target by a Manager agent. The tool wraps the agent's run() method and uses the agent's description for semantic routing.

Returns: A Tool object that wraps this agent.

Method `create_session()`**:** Create a stateful ChatSession from this agent.

Usage:

```
Agent$create_session(model = NULL, ...)
```

Arguments:

`model` Optional model override.

`...` Additional arguments passed to ChatSession$new.

Returns: A ChatSession object initialized with this agent's config.

Method `print()`**:** Print method for Agent.

Usage:

```
Agent$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`agent_evals` *Performance & Benchmarking: Agent Evals*

Description

Testing infrastructure for LLM-powered code. Provides testthat integration with custom expectations for evaluating AI agent performance, tool accuracy, and hallucination rates.

agent_library *Agent Library: Built-in Agent Specialists*

Description

Factory functions for creating standard library agents for common tasks. These agents are pre-configured with appropriate system prompts and tools for their respective specializations.

agent_registry *Agent Registry: Agent Storage and Lookup*

Description

AgentRegistry R6 class for storing and retrieving Agent instances. Used by the Flow system for agent delegation.

AgentRegistry *AgentRegistry Class*

Description

R6 class for managing a collection of Agent objects. Provides storage, lookup, and automatic delegation tool generation for multi-agent systems.

Methods

Public methods:

- `AgentRegistry$new()`
- `AgentRegistry$register()`
- `AgentRegistry$get()`
- `AgentRegistry$has()`
- `AgentRegistry$list_agents()`
- `AgentRegistry$get_all()`
- `AgentRegistry$unregister()`
- `AgentRegistry$generate_delegate_tools()`
- `AgentRegistry$generate_prompt_section()`
- `AgentRegistry$print()`
- `AgentRegistry$clone()`

Method `new()`**:** Initialize a new AgentRegistry.

Usage:

```
AgentRegistry$new(agents = NULL)
```

Arguments:

`agents` Optional list of Agent objects to register immediately.

Method `register()`**:** Register an agent.

Usage:

```
AgentRegistry$register(agent)
```

Arguments:

`agent` An Agent object to register.

Returns: Invisible self for chaining.

Method `get()`**:** Get an agent by name.

Usage:

```
AgentRegistry$get(name)
```

Arguments:

`name` The agent name.

Returns: The Agent object, or NULL if not found.

Method `has()`**:** Check if an agent is registered.

Usage:

```
AgentRegistry$has(name)
```

Arguments:

`name` The agent name.

Returns: TRUE if registered, FALSE otherwise.

Method `list_agents()`**:** List all registered agent names.

Usage:

```
AgentRegistry$list_agents()
```

Returns: Character vector of agent names.

Method `get_all()`**:** Get all registered agents.

Usage:

```
AgentRegistry$get_all()
```

Returns: List of Agent objects.

Method `unregister()`**:** Unregister an agent.

Usage:

```
AgentRegistry$unregister(name)
```

Arguments:

`name` The agent name to remove.

Returns: Invisible self for chaining.

Method `generate_delegate_tools()`**:** Generate delegation tools for all registered agents.

Usage:

```
AgentRegistry$generate_delegate_tools(
  flow = NULL,
  session = NULL,
  model = NULL
)
```

Arguments:

`flow` Optional Flow object for context-aware execution.

`session` Optional ChatSession for shared state.

`model` Optional model ID for agent execution.

Details: Creates a list of Tool objects that wrap each agent's run() method. These tools can be given to a Manager agent for semantic routing.

Returns: A list of Tool objects.

Method `generate_prompt_section()`**:** Generate a prompt section describing available agents.

Usage:

```
AgentRegistry$generate_prompt_section()
```

Details: Creates a formatted string listing all agents and their descriptions. Useful for injecting into a Manager's system prompt.

Returns: A character string.

Method `print()`**:** Print method for AgentRegistry.

Usage:

```
AgentRegistry$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`analyze_image` *Analyze Image*

Description

Analyze an image with a multimodal language model.

Usage

```
analyze_image(model, image, prompt, system = NULL, registry = NULL, ...)
```

Arguments

model	A LanguageModelV1 object or provider:model string.
image	Image source accepted by input_image().
prompt	Prompt describing the analysis task.
system	Optional system prompt.
registry	Optional provider registry.
...	Additional arguments passed to generate_text().

Value

A GenerateResult.

AnthropicProvider *Anthropic Provider Class*

Description

Provider class for Anthropic. Can create language models.

Public fields

specification_version Provider spec version.

Methods

Public methods:

- AnthropicProvider$new()
- AnthropicProvider$enable_caching()
- AnthropicProvider$language_model()
- AnthropicProvider$clone()

Method new(): Initialize the Anthropic provider.

Usage:

```
AnthropicProvider$new(
  api_key = NULL,
  base_url = NULL,
  api_version = NULL,
  headers = NULL,
  name = NULL,
  timeout_seconds = NULL,
  total_timeout_seconds = NULL,
  first_byte_timeout_seconds = NULL,
  connect_timeout_seconds = NULL,
  idle_timeout_seconds = NULL
)
```

Arguments:

`api_key` Anthropic API key. Defaults to ANTHROPIC_API_KEY env var.

`base_url` Base URL for API calls. Defaults to https://api.anthropic.com/v1.

`api_version` Anthropic API version header. Defaults to "2023-06-01".

`headers` Optional additional headers.

`name` Optional provider name override.

`timeout_seconds` Legacy alias for `total_timeout_seconds`.

`total_timeout_seconds` Optional total request timeout in seconds for API calls.

`first_byte_timeout_seconds` Optional time-to-first-byte timeout in seconds for API calls.

`connect_timeout_seconds` Optional connection-establishment timeout in seconds for API calls.

`idle_timeout_seconds` Optional stall timeout in seconds for API calls.

Method `enable_caching()`**:** Enable or disable prompt caching.

Usage:

```
AnthropicProvider$enable_caching(enable = TRUE)
```

Arguments:

`enable` Logical.

Method `language_model()`**:** Create a language model.

Usage:

```
AnthropicProvider$language_model(model_id = "claude-sonnet-4-20250514")
```

Arguments:

`model_id` The model ID (e.g., "claude-sonnet-4-20250514", "claude-3-5-sonnet-20241022").

Returns: An AnthropicLanguageModel object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`api_diagnostics` *API Diagnostics*

Description

Provides diagnostic tools to test internet connectivity, DNS resolution, and API reachability.

as_preview_text *Render a Compact Preview String*

Description

Convert common vector-like and tabular objects into a short text preview for summaries and inspection output.

Usage

```
as_preview_text(x, max_items = 5)
```

Arguments

x	Object to preview.
max_items	Maximum number of items or rows to include.

Value

A character string preview or NULL when no preview is available.

ask_ai *Ask aisdk About Recent R Context*

Description

Collect recent R error context, active script context, session information, and workspace object summaries, then open console_chat() with that context as the initial prompt.

Collects the recent R error/session context and opens console_chat() with that context as the first user message. In RStudio, this function also reads the active source document when available, making it suitable as an Addin binding.

Usage

```
ask_ai(
  prompt = NULL,
  model = NULL,
  skills = "auto",
  skill = NULL,
  context = NULL,
  startup_dir = getwd(),
  working_dir = tempdir(),
  sandbox_mode = "permissive",
  stream = TRUE,
  verbose = FALSE,
```

```
  show_context = FALSE,
  max_context_chars = Inf,
  max_error_age_secs = 300,
  confirm_stale_errors = TRUE,
  ...
)
```

Arguments

prompt	Optional user instruction to add above the collected context.
model	Optional model string, LanguageModelV1, or ChatSession.
skills	Skill paths, "auto", or a SkillRegistry passed to console_chat().
skill	Optional skill name to force for the initial turn.
context	Optional additional context text, or an aisdk_ai_context object to reuse.
startup_dir	R session startup directory for project-aware context.
working_dir	Working directory for sandboxed console tools.
sandbox_mode	Sandbox mode for the console agent.
stream	Whether to stream model output.
verbose	Whether to show debug console output.
show_context	If TRUE, print and return the initial prompt without launching console_chat().
max_context_chars	Maximum formatted context characters. Defaults to Inf, meaning no explicit truncation.
max_error_age_secs	Maximum age in seconds for errors/warnings to be included. Defaults to 300 (5 minutes). Set to Inf to include all errors regardless of age.
confirm_stale_errors	If TRUE (default), show a warning and prompt for confirmation when errors/warnings are detected but appear stale.
...	Additional arguments passed to collect_ai_context().

Value

Invisibly returns the ChatSession from console_chat(), or a preview list when show_context = TRUE.

---

auth_hook *Human-in-the-Loop Authorization*

Description

Provides dynamic authorization hooks to pause Agent execution and request user permission for elevated risk operations.

auto_fix *Autonomous Data Science Pipelines*

Description

Self-healing runtime for R code execution. Implements a "Hypothesis-Fix-Verify" loop that feeds error messages, stack traces, and context back to an LLM for automatic error correction.

Execute R code with automatic error recovery using LLM assistance. When code fails, the error is analyzed and a fix is attempted automatically.

Usage

```
auto_fix(
  expr,
  model = NULL,
  max_attempts = 3,
  context = NULL,
  verbose = TRUE,
  memory = NULL
)
```

Arguments

expr	The R expression to execute.
model	The LLM model to use for error analysis (default: from options).
max_attempts	Maximum number of fix attempts (default: 3).
context	Optional additional context about the code's purpose.
verbose	Print progress messages (default: TRUE).
memory	Optional ProjectMemory object for learning from past fixes.

Value

The result of successful execution, or an error if all attempts fail.

Examples

```
## Not run:
# Simple usage - auto-fix a data transformation
result <- auto_fix({
  df <- read.csv("data.csv")
  df %>%
    filter(value > 100) %>%
    summarize(mean = mean(value))
})

# With context for better error understanding
```

```
result <- auto_fix(
  expr = {
    model <- lm(y ~ x, data = df)
  },
  context = "Fitting a linear regression model to predict sales"
)

## End(Not run)
```

---

cache *Caching System*

Description

Utilities for caching tool execution results and other expensive operations.

---

cache_tool *Cache Tool*

Description

Wrap a tool with caching capabilities using the memoise package.

Usage

```
cache_tool(tool, cache = NULL)
```

Arguments

tool	The Tool object to cache.
cache	An optional memoise cache configuration (e.g., cache_memory() or cache_filesystem()). Defaults to memoise::cache_memory().

Value

A new Tool object that caches its execution.

call_object_accessor *Call an Object Accessor by Candidate Function Names*

Description

Try accessor functions in order and return the first successful result. Useful for extension authors who need compatibility across optional dependency APIs.

Usage

```
call_object_accessor(
  obj,
  fun_names,
  default = NULL,
  package = NULL,
  args = list()
)
```

Arguments

obj	Object passed as the first argument to the accessor.
fun_names	Character vector of accessor function names to try.
default	Value returned when no accessor can be called successfully.
package	Optional package name to resolve accessors from first.
args	Optional named list of additional arguments passed to accessor.

Value

The accessor result or default.

capability_models *Capability Model Routes*

Description

Configure model routes for task capabilities such as vision inspection, image generation, or code review. These routes let a session keep one default chat model while specific capabilities use better-suited models.

ChatSession *ChatSession Class*

Description

R6 class representing a stateful chat session. Automatically manages conversation history, supports tool execution, and provides persistence.

Methods

Public methods:

- ChatSession$new()
- ChatSession$send()
- ChatSession$send_stream()
- ChatSession$continue_run()
- ChatSession$append_message()
- ChatSession$get_history()
- ChatSession$get_last_response()
- ChatSession$clear_history()
- ChatSession$switch_model()
- ChatSession$get_model_options()
- ChatSession$get_model_call_options()
- ChatSession$set_model_options()
- ChatSession$clear_model_options()
- ChatSession$set_capability_model()
- ChatSession$get_capability_model()
- ChatSession$list_capability_models()
- ChatSession$clear_capability_model()
- ChatSession$get_model_id()
- ChatSession$get_model()
- ChatSession$get_tools()
- ChatSession$stats()
- ChatSession$save()
- ChatSession$as_list()
- ChatSession$restore()
- ChatSession$restore_from_list()
- ChatSession$print()
- ChatSession$get_memory()
- ChatSession$get_run_state()
- ChatSession$set_run_state()
- ChatSession$set_memory()
- ChatSession$list_memory()

- ChatSession$get_metadata()
- ChatSession$set_metadata()
- ChatSession$merge_metadata()
- ChatSession$list_metadata()
- ChatSession$get_context_state()
- ChatSession$set_context_state()
- ChatSession$clear_context_state()
- ChatSession$get_context_management_mode()
- ChatSession$get_context_management_config()
- ChatSession$set_context_management_mode()
- ChatSession$set_context_management_config()
- ChatSession$get_context_metrics()
- ChatSession$assemble_messages()
- ChatSession$refresh_context_state()
- ChatSession$list_context_handles()
- ChatSession$create_context_query_tools()
- ChatSession$sub_session_query()
- ChatSession$clear_memory()
- ChatSession$get_envir()
- ChatSession$eval_in_session()
- ChatSession$list_envir()
- ChatSession$checkpoint()
- ChatSession$restore_checkpoint()
- ChatSession$clone()

Method `new()`**:** Initialize a new ChatSession.

Usage:

```
ChatSession$new(
  model = NULL,
  system_prompt = NULL,
  tools = NULL,
  hooks = NULL,
  history = NULL,
  max_steps = 10,
  registry = NULL,
  memory = NULL,
  metadata = NULL,
  envir = NULL,
  agent = NULL
)
```

Arguments:

`model` A LanguageModelV1 object or model string ID (e.g., "openai:gpt-4o").

`system_prompt` Optional system prompt for the conversation.

`tools` Optional list of Tool objects for function calling.

`hooks` Optional HookHandler object for event hooks.

`history` Optional initial message history (list of message objects).

`max_steps` Maximum steps for tool execution loops. Default 10.

`registry` Optional ProviderRegistry for model resolution.

`memory` Optional initial shared memory (list). For multi-agent state sharing.

`metadata` Optional session metadata (list). Used for channel/runtime state.

`envir` Optional shared R environment. For multi-agent data sharing.

`agent` Optional Agent object. If provided, the session inherits the agent's tools and system prompt.

Method `send()`**:** Send a message and get a response.

Usage:

```
ChatSession$send(prompt, ...)
```

Arguments:

`prompt` The user message to send.

`...` Additional arguments passed to generate_text.

Returns: The GenerateResult object from the model.

Method `send_stream()`**:** Send a message with streaming output.

Usage:

```
ChatSession$send_stream(prompt, callback, ...)
```

Arguments:

`prompt` The user message to send.

`callback` Function called for each chunk: callback(text, done).

`...` Additional arguments passed to stream_text.

Returns: The GenerateResult object invisibly (output is via callback).

Method `continue_run()`**:** Inject a manual continuation instruction into the current task.

Usage:

```
ChatSession$continue_run(
  action = "continue",
  guidance = NULL,
  stream = TRUE,
  callback = NULL,
  ...
)
```

Arguments:

`action` One of "continue", "give_up", "avoid_tool", "explain", or "manual".

`guidance` Optional operator guidance to include in the continuation.

`stream` Whether to use streaming generation.

`callback` Streaming callback when `stream = TRUE`.

`...` Additional arguments passed to send/send_stream.

Returns: The GenerateResult object, or an invisible waiting-user result for manual action.

Method `append_message()`**:** Append a message to the history.

Usage:

```
ChatSession$append_message(role, content, reasoning = NULL)
```

Arguments:

`role` Message role: "user", "assistant", "system", or "tool".

`content` Message content.

`reasoning` Optional reasoning text to attach to the message.

Method `get_history()`**:** Get the conversation history.

Usage:

```
ChatSession$get_history()
```

Returns: A list of message objects.

Method `get_last_response()`**:** Get the last response from the assistant.

Usage:

```
ChatSession$get_last_response()
```

Returns: The content of the last assistant message, or NULL.

Method `clear_history()`**:** Clear the conversation history.

Usage:

```
ChatSession$clear_history(keep_system = TRUE)
```

Arguments:

`keep_system` If TRUE, keeps the system prompt. Default TRUE.

Method `switch_model()`**:** Switch to a different model.

Usage:

```
ChatSession$switch_model(model)
```

Arguments:

`model` A LanguageModelV1 object or model string ID.

Method `get_model_options()`**:** Get model runtime options for this session.

Usage:

```
ChatSession$get_model_options()
```

Returns: A list with context overrides and call options.

Method `get_model_call_options()`**:** Get generation options applied to every model call.

Usage:

```
ChatSession$get_model_call_options()
```

Returns: A named list of call options.

Method `set_model_options()`**:** Set runtime options for this session's model.

Usage:

```
ChatSession$set_model_options(
  context_window = NULL,
  max_output_tokens = NULL,
  max_tokens = NULL,
  thinking = NULL,
  thinking_budget = NULL,
  reasoning_effort = NULL,
  reset = FALSE
)
```

Arguments:

`context_window` Optional context-window override.

`max_output_tokens` Optional maximum output-token metadata override.

`max_tokens` Optional default generation token limit.

`thinking` Optional default thinking-mode value.

`thinking_budget` Optional default thinking budget.

`reasoning_effort` Optional default reasoning effort.

`reset` Logical. If TRUE, clears all model runtime options first.

Returns: Invisible self for chaining.

Method `clear_model_options()`**:** Clear model runtime options for this session.

Usage:

```
ChatSession$clear_model_options(keys = NULL)
```

Arguments:

`keys` Optional option names to clear. If NULL, clears all.

Returns: Invisible self for chaining.

Method `set_capability_model()`**:** Set a model route for a session capability.

Usage:

```
ChatSession$set_capability_model(
  capability,
  model,
  type = "auto",
  required_model_capabilities = NULL
)
```

Arguments:

`capability` Capability route name, such as "vision.inspect".

`model` Model ID string or model object. Passing NULL clears the route.

`type` Model type: "auto", "language", "embedding", or "image".

`required_model_capabilities` Optional required model capability flags.

Returns: Invisible self for chaining.

Method `get_capability_model()`**:** Get the configured model for a session capability.

Usage:

```
ChatSession$get_capability_model(capability, default = NULL)
```

Arguments:

`capability` Capability route name.

`default` Value returned when no route is configured.

Returns: A model ID string, model object, or default.

Method `list_capability_models()`**:** List session capability model routes.

Usage:

```
ChatSession$list_capability_models()
```

Returns: A data frame of configured session routes.

Method `clear_capability_model()`**:** Clear one or all session capability model routes.

Usage:

```
ChatSession$clear_capability_model(capability = NULL)
```

Arguments:

`capability` Optional route name. If NULL, clears all routes.

Returns: Invisible self for chaining.

Method `get_model_id()`**:** Get current model identifier.

Usage:

```
ChatSession$get_model_id()
```

Returns: Model ID string.

Method `get_model()`**:** Get the resolved language model for this session.

Usage:

```
ChatSession$get_model()
```

Returns: A `LanguageModelV1` object.

Method `get_tools()`**:** Get tools configured on this session.

Usage:

```
ChatSession$get_tools()
```

Returns: A list of `Tool` objects.

Method `stats()`**:** Get token usage statistics.

Usage:

```
ChatSession$stats()
```

Returns: A list with token counts and message stats.

Method `save()`**:** Save session to a file.

Usage:

```
ChatSession$save(path, format = NULL)
```

Arguments:

path File path (supports .rds or .json extension).
format Optional format override: "rds" or "json". Auto-detected from path.

Method `as_list()`**:** Export session state as a list (for serialization).

Usage:

```
ChatSession$as_list()
```

Returns: A list containing session state.

Method `restore()`**:** Restore session from a file.

Usage:

```
ChatSession$restore(path, format = NULL)
```

Arguments:

path File path (supports .rds or .json extension).
format Optional format override: "rds" or "json". Auto-detected from path.

Method `restore_from_list()`**:** Restore session state from a list.

Usage:

```
ChatSession$restore_from_list(data)
```

Arguments:

data A list exported by as_list().

Method `print()`**:** Print method for ChatSession.

Usage:

```
ChatSession$print()
```

Method `get_memory()`**:** Get a value from shared memory.

Usage:

```
ChatSession$get_memory(key, default = NULL)
```

Arguments:

key The key to retrieve.
default Default value if key not found. Default NULL.

Returns: The stored value or default.

Method `get_run_state()`**:** Return the most recent structured run state.

Usage:

```
ChatSession$get_run_state()
```

Returns: A run state list.

Method `set_run_state()`**:** Store the current structured run state.

Usage:

```
ChatSession$set_run_state(run_state)
```

Arguments:

run_state A run state list.

Returns: Invisible self.

Method `set_memory()`**:** Set a value in shared memory.

Usage:

```
ChatSession$set_memory(key, value)
```

Arguments:

`key` The key to store.

`value` The value to store.

Returns: Invisible self for chaining.

Method `list_memory()`**:** List all keys in shared memory.

Usage:

```
ChatSession$list_memory()
```

Returns: Character vector of memory keys.

Method `get_metadata()`**:** Get a value from session metadata.

Usage:

```
ChatSession$get_metadata(key, default = NULL)
```

Arguments:

`key` The metadata key to retrieve.

`default` Default value if key is not present.

Returns: The stored metadata value or default.

Method `set_metadata()`**:** Set a value in session metadata.

Usage:

```
ChatSession$set_metadata(key, value)
```

Arguments:

`key` The metadata key to set.

`value` The value to store.

Returns: Invisible self for chaining.

Method `merge_metadata()`**:** Merge a named list into session metadata.

Usage:

```
ChatSession$merge_metadata(values)
```

Arguments:

`values` Named list of metadata values.

Returns: Invisible self for chaining.

Method `list_metadata()`**:** List metadata keys.

Usage:

```
ChatSession$list_metadata()
```

Returns: Character vector of metadata keys.

Method `get_context_state()`**:** Get the adaptive context state for this session.

Usage:

```
ChatSession$get_context_state()
```

Returns: A normalized context state list.

Method `set_context_state()`**:** Store adaptive context state for this session.

Usage:

```
ChatSession$set_context_state(state)
```

Arguments:

`state` Context state list.

Returns: Invisible self for chaining.

Method `clear_context_state()`**:** Clear adaptive context state back to defaults.

Usage:

```
ChatSession$clear_context_state()
```

Returns: Invisible self for chaining.

Method `get_context_management_mode()`**:** Get the context management mode for this session.

Usage:

```
ChatSession$get_context_management_mode()
```

Returns: One of "off", "basic", or "adaptive".

Method `get_context_management_config()`**:** Get the full adaptive context-management configuration.

Usage:

```
ChatSession$get_context_management_config()
```

Returns: A normalized context-management configuration list.

Method `set_context_management_mode()`**:** Override the context management mode for this session.

Usage:

```
ChatSession$set_context_management_mode(mode)
```

Arguments:

`mode` One of "off", "basic", or "adaptive".

Returns: Invisible self for chaining.

Method `set_context_management_config()`**:** Apply adaptive context-management configuration to this session.

Usage:

```
ChatSession$set_context_management_config(config = NULL, ...)
```

Arguments:

`config` Optional config list created by `create_context_management_config()`.

`...` Optional overrides forwarded to `set_context_management_config()`.

Returns: Invisible self for chaining.

Method `get_context_metrics()`**:** Estimate current context metrics for this session.

Usage:

```
ChatSession$get_context_metrics(turn_system_prompt = NULL)
```

Arguments:

`turn_system_prompt` Optional turn-specific system prompt to include in the estimate.

Returns: A list of context metrics, or NULL if no model metadata is available.

Method `assemble_messages()`**:** Build a budget-aware prompt payload from current session history.

Usage:

```
ChatSession$assemble_messages(turn_system_prompt = NULL)
```

Arguments:

`turn_system_prompt` Optional turn-specific system prompt.

Returns: A list with `messages`, `system`, `metrics`, and `state`.

Method `refresh_context_state()`**:** Refresh the adaptive context state from current history.

Usage:

```
ChatSession$refresh_context_state(
  generation_result = NULL,
  turn_system_prompt = NULL
)
```

Arguments:

`generation_result` Optional GenerateResult used to update tool/artifact digests.

`turn_system_prompt` Optional turn-specific system prompt for the snapshot.

Returns: The normalized context state list.

Method `list_context_handles()`**:** List compact context handles available to this session.

Usage:

```
ChatSession$list_context_handles()
```

Returns: A list of context handle records.

Method `create_context_query_tools()`**:** Create context query tools bound to this session.

Usage:

```
ChatSession$create_context_query_tools()
```

Returns: A list of `Tool` objects.

Method `sub_session_query()`**:** Run a bounded child session for a focused query.

Usage:

```
ChatSession$sub_session_query(...)
```

Arguments:

`...` Arguments passed to `sub_session_query()`.

Returns: A compact sub-session result list.

Method `clear_memory()`**:** Clear shared memory.

Usage:

```
ChatSession$clear_memory(keys = NULL)
```

Arguments:

`keys` Optional specific keys to clear. If NULL, clears all.

Returns: Invisible self for chaining.

Method `get_envir()`**:** Get the shared R environment.

Usage:

```
ChatSession$get_envir()
```

Details: This environment is shared across all agents using this session. Agents can store and retrieve data frames, models, and other R objects.

Returns: An environment object.

Method `eval_in_session()`**:** Evaluate an expression in the session environment.

Usage:

```
ChatSession$eval_in_session(expr)
```

Arguments:

`expr` An expression to evaluate.

Returns: The result of the evaluation.

Method `list_envir()`**:** List objects in the session environment.

Usage:

```
ChatSession$list_envir()
```

Returns: Character vector of object names.

Method `checkpoint()`**:** Save a memory snapshot to a file (checkpoint for Mission resume).

Usage:

```
ChatSession$checkpoint(path = NULL)
```

Arguments:

`path` File path (.rds). If NULL, uses a temp file and returns the path.

Returns: Invisible file path.

Method `restore_checkpoint()`**:** Restore memory and history from a checkpoint file.

Usage:

```
ChatSession$restore_checkpoint(path)
```

Arguments:

`path` File path to a checkpoint created by checkpoint().

Returns: Invisible self for chaining.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`check_api` *Connect and Diagnose API Reachability*

Description

Tests connectivity to a specific LLM, provider, or URL. This is helpful for diagnosing network issues, DNS failures, or SSL problems.

Usage

```
check_api(model = NULL, url = NULL, registry = NULL)
```

Arguments

`model`	Optional. A `LanguageModelV1` object, or a provider name string (e.g., "openai", "gemini"). If provided, the base URL for that provider will be tested.
`url`	Optional. A specific URL to test.
`registry`	Optional ProviderRegistry to use if `model` is a character string.

Value

A list containing diagnostic results (invisible).

Examples

```
if (interactive()) {
    # Test by passing a URL directly
    check_api(url = "https://api.openai.com/v1")

    # Test a model directly
    model <- create_openai()$language_model("gpt-4o")
    check_api(model)
}
```

check_ast_safety *Check AST Safety*

Description

Analyze R code for unsafe function calls or operations before execution.

Usage

```
check_ast_safety(code_str)
```

Arguments

code_str	Character string containing R code.

Value

The parsed AST if safe. Throws an error if unsafe.

check_sdk_compatibility *Check SDK Version Compatibility*

Description

Check if code is compatible with the current SDK version and suggest migration steps if needed.

Usage

```
check_sdk_compatibility(code_version)
```

Arguments

code_version	Version string the code was written for.

Value

A list with compatible (logical) and suggestions (character vector).

Examples

```
if (interactive()) {
result <- check_sdk_compatibility("0.8.0")
if (!result$compatible) {
  cat("Migration needed:\n")
  cat(paste(result$suggestions, collapse = "\n"))
}
}
```

clear_capability_model *Clear Capability Model*

Description

Clear one or all configured capability model routes.

Usage

```
clear_capability_model(capability = NULL)
```

Arguments

capability	Optional capability route name. If NULL, all routes are cleared.

Value

Invisibly returns the previous route list.

clear_error_context *Clear Error Context for ask_ai()*

Description

Clears the tracked error and warning context used by ask_ai(). This is useful when you want to start a fresh debugging session without stale error messages from previous operations.

Usage

```
clear_error_context()
```

Value

Invisibly returns TRUE.

Examples

```
## Not run:
# Clear stale errors before starting a new debugging session
clear_error_context()
ask_ai()

## End(Not run)
```

collect_ai_context *Collect Context for* ask_ai()

Description

Collect recent error details, traceback output, warnings, active script context, session information, and lightweight workspace object summaries.

Usage

```
collect_ai_context(
  script = NULL,
  error = NULL,
  traceback = NULL,
  warnings = NULL,
  include = c("error", "traceback", "warnings", "script", "session", "objects",
    "history"),
  max_context_chars = Inf,
  max_error_age_secs = 300,
  include_history = TRUE,
  max_history_lines = 10
)
```

Arguments

script	Optional script text or path. If NULL, RStudio active document context is used when available.
error	Optional error message. Defaults to geterrmessage().
traceback	Optional traceback text. Defaults to traceback() output.
warnings	Optional warning text or warning object. Defaults to R's last.warning context when present.
include	Character vector of sections to include.
max_context_chars	Maximum formatted context characters. Defaults to Inf, meaning no explicit truncation.
max_error_age_secs	Maximum age in seconds for errors/warnings to be included. Defaults to 300 (5 minutes). Set to Inf to include all errors.
include_history	Whether to include recent command history. Defaults to TRUE.
max_history_lines	Maximum number of recent command history lines to include. Defaults to 10.

Value

A structured context list with class aisdk_ai_context.

compat *Compatibility Layer: Feature Flags and Migration Support*

Description

Provides feature flags, compatibility shims, and migration utilities for controlled breaking changes in the agent SDK.

Computer *Computer Class*

Description

R6 class providing computer abstraction with atomic tools for file operations, bash execution, and R code execution.

Public fields

`working_dir` Current working directory

`sandbox_mode` Sandbox mode: "strict", "permissive", or "none"

`execution_log` Log of executed commands

Methods

Public methods:

- `Computer$new()`
- `Computer$bash()`
- `Computer$read_file()`
- `Computer$write_file()`
- `Computer$edit_file()`
- `Computer$execute_r_code()`
- `Computer$get_log()`
- `Computer$clear_log()`
- `Computer$clone()`

Method `new()`**:** Initialize computer abstraction

Usage:

```
Computer$new(working_dir = tempdir(), sandbox_mode = "permissive")
```

Arguments:

`working_dir` Working directory. Defaults to `tempdir()`.

`sandbox_mode` Sandbox mode: "strict", "permissive", or "none"

Method `bash()`: Execute bash command

Usage:

```
Computer$bash(command, timeout_ms = 30000, capture_output = TRUE)
```

Arguments:

`command` Bash command to execute
`timeout_ms` Timeout in milliseconds (default: 30000)
`capture_output` Whether to capture output (default: TRUE)

Returns: List with stdout, stderr, exit_code

Method `read_file()`: Read file contents

Usage:

```
Computer$read_file(path, encoding = "UTF-8")
```

Arguments:

`path` File path (relative to working_dir or absolute)
`encoding` File encoding (default: "UTF-8")

Returns: File contents as character string

Method `write_file()`: Write file contents

Usage:

```
Computer$write_file(path, content, encoding = "UTF-8")
```

Arguments:

`path` File path (relative to working_dir or absolute)
`content` Content to write
`encoding` File encoding (default: "UTF-8")

Returns: Success status

Method `edit_file()`: Edit a file by replacing an exact text pattern.

Usage:

```
Computer$edit_file(path, pattern, replacement, all = FALSE, encoding = "UTF-8")
```

Arguments:

`path` File path (relative to working_dir or absolute)
`pattern` Exact text to replace
`replacement` Replacement text
`all` Whether to replace all occurrences. Default FALSE.
`encoding` File encoding (default: "UTF-8")

Returns: Success status

Method `execute_r_code()`: Execute R code in an isolated `callr` process

Usage:

```
Computer$execute_r_code(code, timeout_ms = 30000, capture_output = TRUE)
```

Arguments:

code R code to execute

timeout_ms Timeout in milliseconds (default: 30000)

capture_output Whether to capture output (default: TRUE)

Returns: List with result, output, error, and execution_mode. execution_mode is always "sandbox_exec" for this computer-layer path, which does not persist values into a live ChatSession$get_envir().

Method get_log(): Get execution log

Usage:

```
Computer$get_log()
```

Returns: List of logged executions

Method clear_log(): Clear execution log Check bash command for sandbox violations Log execution Resolve path (relative to working_dir or absolute) Check write path for sandbox violations Check R code for sandbox violations

Usage:

```
Computer$clear_log()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

---

config_model *Model Configuration Files*

Description

Helpers for reading project/global aisdk.yaml model configuration files.

---

console *Console Chat: Interactive REPL*

Description

Interactive terminal chat interface for ChatSession. Provides a REPL (Read-Eval-Print Loop) for conversing with LLMs. By default, enables an intelligent terminal agent that can execute commands, manage files, and run R code through natural language.

console_agent *Console Agent: Intelligent Terminal Assistant*

Description

Creates a default agent for console_chat() that enables natural language interaction with the terminal. Users can ask the agent to run commands, execute R code, read/write files, and more through conversational language.

console_chat *Start Console Chat*

Description

Launch an interactive chat session in the R console. Supports streaming output, slash commands, and colorful display using the cli package.

The console UI has three presentation modes:

- `clean`: compact default output with a stable status bar
- `inspect`: keeps the compact transcript but adds a per-turn tool timeline and an overlay-backed inspector
- `debug`: shows detailed tool logs and thinking output for troubleshooting

In agent mode, `console_chat()` can execute shell and R tools, summarize tool progress inline, and open an inspector overlay for the latest turn or a specific tool. The current implementation uses a shared frame builder for the status bar, tool timeline, and overlay surfaces, while preserving an append-only terminal fallback.

By default, the console operates in minimal agent mode with four tools: `bash`, `read_file`, `write_file`, and `edit_file`. Set `profile = "legacy"` to restore the previous broad all-in-one agent, or `agent = NULL` for simple chat without tools.

Usage

```
console_chat(
  session = NULL,
  system_prompt = NULL,
  tools = NULL,
  hooks = NULL,
  stream = TRUE,
  verbose = FALSE,
  agent = "auto",
  skills = NULL,
  working_dir = tempdir(),
  sandbox_mode = "permissive",
```

```
  show_thinking = verbose,
  startup_dir = getwd(),
  initial_prompt = NULL,
  profile = c("minimal", "legacy"),
  extensions = "auto"
)
```

Arguments

session	A ChatSession object, a LanguageModelV1 object, or a model string ID to create a new session.
system_prompt	Optional system prompt (merged with agent prompt if agent is used).
tools	Optional list of additional Tool objects.
hooks	Optional HookHandler object.
stream	Whether to use streaming output. Default TRUE.
verbose	Logical. If TRUE, show detailed tool calls, tool results, and thinking output. Defaults to FALSE for a cleaner console UI.
agent	Agent configuration. Options: • "auto" (default): Use the built-in console agent with terminal tools • NULL: Simple chat mode without tools • An Agent object: Use the provided custom agent
skills	Optional skill paths, "auto", or a SkillRegistry object for the built-in console agent. Defaults to automatic skill discovery when agent = "auto".
working_dir	Working directory for sandboxed console tools. Defaults to tempdir().
sandbox_mode	Sandbox mode for the console agent: "strict", "permissive" (default), or "none".
show_thinking	Logical. Whether to show model thinking blocks when the provider exposes them. Defaults to verbose.
startup_dir	R session startup directory used for project-aware context. Defaults to getwd().
initial_prompt	Optional user prompt to send automatically before entering the interactive REPL.
profile	Console profile. "minimal" is the default Pi-like tool set; "legacy" restores the previous all-in-one console agent.
extensions	Extension loading mode. Defaults to "auto".

Value

The ChatSession object (invisibly) when chat ends.

Examples

```
if (interactive()) {
  # Start with default agent (intelligent terminal mode)
  console_chat("openai:gpt-4o")

  # Start in debug mode with full tool logs
  console_chat("openai:gpt-4o", verbose = TRUE)
```

```
  # Simple chat mode without tools
  console_chat("openai:gpt-4o", agent = NULL)

  # Start with an existing session
  chat <- create_chat_session("anthropic:claude-3-5-sonnet-latest")
  console_chat(chat)

  # Start with a custom agent
  agent <- create_agent("MathAgent", "Does math", system_prompt = "You are a math wizard.")
  console_chat("openai:gpt-4o", agent = agent)

  # Available commands in the chat:
  # /quit or /exit - End the chat
  # /save [path]   - Save session to file
  # /load [path]   - Load session from file
  # /model         - Open the provider/model chooser
  # /model [id]    - Switch to a different model
  # /model current - Show the active model
  # /history       - Show conversation history
  # /stats         - Show token usage statistics
  # /clear         - Clear conversation history
  # /stream [on|off] - Toggle streaming mode
  # /inspect [on|off] - Toggle inspect mode
  # /inspect turn - Open overlay for the latest turn
  # /inspect tool <index> - Open overlay for a tool in the latest turn
  # /inspect next - Move inspector overlay to the next tool
  # /inspect prev - Move inspector overlay to the previous tool
  # /inspect close - Close the active inspect overlay
  # /debug [on|off]  - Toggle detailed tool/thinking output
  # /local [on|off]- Toggle local execution mode (Global Environment)
  # /help          - Show available commands
  # /agent [on|off] - Toggle agent mode
}
```

`console_confirm` *Console Confirmation Prompt*

Description

Ask a yes/no question with numbered choices. Returns `TRUE` for yes, `FALSE` for no, or `NULL` if cancelled.

Usage

```
console_confirm(question)
```

Arguments

`question`	The question to display.

Value

TRUE if user selects Yes, FALSE for No, NULL if cancelled.

Examples

```
if (interactive()) {
  if (isTRUE(console_confirm("Overwrite existing file?"))) {
    message("Overwriting...")
  }
}
```

---

console_input *Console Text Input*

Description

Prompt the user for free-text input with optional default value.

Usage

```
console_input(prompt, default = NULL)
```

Arguments

prompt	The prompt message to display.
default	Optional default value shown in brackets. Returned if user presses Enter without typing.

Value

The user's input string, default if empty input and default is set, or NULL if empty input with no default.

Examples

```
if (interactive()) {
  name <- console_input("Project name", default = "my-project")
  api_key <- console_input("API key")
}
```

console_menu *Console Interactive Menu*

Description

Present a numbered list of choices and return the user's selection. Styled with cli to match the console chat interface. Similar to `utils::menu()` but with cli formatting.

Usage

```
console_menu(title, choices)
```

Arguments

`title`	The question or prompt to display.
`choices`	Character vector of options to present.

Value

The index of the selected choice (integer), or `NULL` if cancelled (user enters 'q' or empty input).

Examples

```
if (interactive()) {
  selection <- console_menu("Which database?", c("PostgreSQL", "SQLite", "DuckDB"))
}
```

content_blocks *Provider-Neutral Content Blocks*

Description

Helpers for constructing and validating provider-neutral multimodal content blocks that can later be translated into provider-specific payloads.

content_image *Create Image Content*

Description

Creates an image content object for a multimodal message. Automatically handles URLs, data URIs, raw bytes, and local files. This is kept for backward compatibility and returns the provider-neutral image block used by multimodal providers.

Usage

```
content_image(image_path, media_type = "auto", detail = "auto")
```

Arguments

image_path	Path to a local file or a URL.
media_type	MIME type of the image (e.g., "image/jpeg", "image/png"). If NULL, attempts to guess from the file extension.
detail	Image detail suitable for some models (e.g., "auto", "low", "high").

Value

A provider-neutral image block.

content_text *Create Text Content*

Description

Creates a text content object for a multimodal message. This is kept for backward compatibility and returns the provider-neutral text block used by multimodal providers.

Usage

```
content_text(text)
```

Arguments

text	The text string.

Value

A list representing the text content.

context *Context Management*

Description

Utilities for capturing and summarizing R objects for LLM context.

context_budget *Adaptive Context Budget Helpers*

Description

Internal helpers for estimating prompt occupancy, classifying context budget regimes, storing compact session-side context state, and assembling a budget-aware prompt view from raw session history.

context_collectors *Context State Collectors*

Description

Internal helpers for collecting execution monitoring, system/device info, and R runtime state signals to populate context state fields.

context_get *Get Session Context by Handle*

Description

Resolve a compact context handle to either its summary metadata or a bounded detailed representation.

Usage

```
context_get(session, handle_id, detail = c("summary", "full"))
```

Arguments

session	A ChatSession or SharedSession.
handle_id	Context handle ID.
detail	One of "summary" or "full".

Value

A list with handle metadata and optional bounded content.

context_management *Context Management Configuration*

Description

Public helpers for configuring adaptive context management on `ChatSession` objects without reaching into internal metadata keys.

context_search *Search Session Context*

Description

Search compact handles for live objects, memory entries, transcript segments, retrieval hits, and artifacts.

Usage

```
context_search(session, query, kinds = NULL, limit = 5L)
```

Arguments

`session`	A `ChatSession` or `SharedSession`.
`query`	Search query.
`kinds`	Optional character vector of handle kinds to include.
`limit`	Maximum number of handles to return.

Value

A list of matching context handles.

core_api *Core API: High-Level Functions*

Description

User-facing high-level API functions for interacting with AI models.

core_object *Core Object API: Structured Output Generation*

Description

Functions for generating structured objects from LLMs using schemas.

Generate a structured R object (list) from a language model based on a schema. The model is instructed to output valid JSON matching the schema, which is then parsed and returned as an R list.

Usage

```
generate_object(
  model = NULL,
  prompt,
  schema,
  schema_name = "result",
  system = NULL,
  temperature = 0.3,
  max_tokens = NULL,
  mode = c("json", "tool"),
  registry = NULL,
  ...
)
```

Arguments

model	Either a LanguageModelV1 object, or a string ID like "openai:gpt-4o".
prompt	A character string prompt describing what to generate.
schema	A schema object created by z_object(), z_array(), etc.
schema_name	Optional human-readable name for the schema (default: "result").
system	Optional system prompt.
temperature	Sampling temperature (0-2). Default 0.3 (lower for structured output).
max_tokens	Maximum tokens to generate.
mode	Output mode: "json" (prompt-based) or "tool" (function calling). Currently, only "json" mode is implemented.
registry	Optional ProviderRegistry to use (defaults to global registry).
...	Additional arguments passed to the model.

Value

A GenerateObjectResult with:

- object: The parsed R object (list)

- usage: Token usage information
- raw_text: The raw text output from the LLM
- finish_reason: The reason the generation stopped

Examples

```
if (interactive()) {
# Define a schema for the expected output
schema <- z_object(
  title = z_string(description = "Title of the article"),
  keywords = z_array(z_string()),
  sentiment = z_enum(c("positive", "negative", "neutral"))
)

# Generate structured object
result <- generate_object(
  model = "openai:gpt-4o",
  prompt = "Analyze this article: 'R programming is great for data science!'",
  schema = schema
)

print(result$object$title)
print(result$object$sentiment)
}
```

`create_agent` *Create an Agent*

Description

Factory function to create a new Agent object.

Usage

```
create_agent(
  name,
  description,
  system_prompt = NULL,
  tools = NULL,
  skills = NULL,
  model = NULL,
  capability_models = NULL
)
```

Arguments

name	Unique name for this agent.
description	A clear description of what this agent does.
system_prompt	Optional system prompt defining the agent's persona.
tools	Optional list of Tool objects the agent can use.
skills	Optional character vector of skill paths or "auto".
model	Optional default model ID for this agent.
capability_models	Optional named list of capability-specific model routes.

Value

An Agent object.

Examples

```
if (interactive()) {
  # Create a simple math agent
  math_agent <- create_agent(
    name = "MathAgent",
    description = "Performs arithmetic calculations",
    system_prompt = "You are a math assistant. Return only numerical results."
  )

  # Run the agent
  result <- math_agent$run("Calculate 2 + 2", model = "openai:gpt-4o")

  # Create an agent with skills
  stock_agent <- create_agent(
    name = "StockAnalyst",
    description = "Stock analysis agent",
    skills = "auto"
  )
}
```

---

create_agent_registry *Create an Agent Registry*

Description

Factory function to create a new AgentRegistry.

Usage

```
create_agent_registry(agents = NULL)
```

Arguments

agents	Optional list of Agent objects to register immediately.

Value

An AgentRegistry object.

Examples

```
if (interactive()) {
# Create registry with agents
cleaner <- create_agent("Cleaner", "Cleans data")
plotter <- create_agent("Plotter", "Creates visualizations")

registry <- create_agent_registry(list(cleaner, plotter))
print(registry$list_agents())  # "Cleaner", "Plotter"
}
```

---

create_anthropic *Create Anthropic Provider*

Description

Factory function to create an Anthropic provider.

Usage

```
create_anthropic(
  api_key = NULL,
  base_url = NULL,
  api_version = NULL,
  headers = NULL,
  name = NULL,
  timeout_seconds = NULL,
  total_timeout_seconds = NULL,
  first_byte_timeout_seconds = NULL,
  connect_timeout_seconds = NULL,
  idle_timeout_seconds = NULL
)
```

Arguments

api_key	Anthropic API key. Defaults to ANTHROPIC_API_KEY env var.
base_url	Base URL for API calls. Defaults to https://api.anthropic.com/v1.
api_version	Anthropic API version header. Defaults to "2023-06-01".
headers	Optional additional headers.

`name` Optional provider name override.

`timeout_seconds`
Legacy alias for `total_timeout_seconds`.

`total_timeout_seconds`
Optional total request timeout in seconds for API calls.

`first_byte_timeout_seconds`
Optional time-to-first-byte timeout in seconds for API calls.

`connect_timeout_seconds`
Optional connection-establishment timeout in seconds for API calls.

`idle_timeout_seconds`
Optional stall timeout in seconds for API calls.

Value

An AnthropicProvider object.

Supported Models

- **claude-opus-4-7**: Anthropic's most capable model for complex reasoning and coding. (Reasoning, Vision, Tools, Structured) | ctx: 1000k
- **claude-opus-4-6**: Previous generation of Claude Opus with 1M context. (Reasoning, Vision, Tools, Structured) | ctx: 1000k
- **claude-opus-4-5**: Claude Opus 4.5 with 200K context. (Reasoning, Vision, Tools, Structured) | ctx: 200k
- **claude-sonnet-4-6**: Balanced model for coding and agentic workflows with 1M context. (Reasoning, Vision, Tools, Structured) | ctx: 1000k
- **claude-sonnet-4-5**: Claude Sonnet 4.5 with 200K context. (Reasoning, Vision, Tools, Structured) | ctx: 200k
- **claude-sonnet-4**: Original Claude Sonnet 4 with 200K context. (Reasoning, Vision, Tools, Structured) | ctx: 200k
- **claude-haiku-4-5**: Fast and cost-effective model for simple tasks and chatbots. (Vision, Tools, Structured) | ctx: 200k

Examples

```
if (interactive()) {
  anthropic <- create_anthropic(api_key = "sk-ant-...")
  model <- anthropic$language_model("claude-sonnet-4-20250514")
}
```

`create_ask_user_tool` *Create an ask_user Tool*

Description

Creates a tool that allows an agent to pause execution and ask the user for help, clarification, or guidance. This is reserved for real user decisions such as destructive actions, credentials, cost/permission gates, ambiguous requirements, or information only the user can provide.

Usage

```
create_ask_user_tool()
```

Details

The ask_user tool lets agents pause for genuine user decisions instead of guessing through risks or missing requirements.

The tool accepts:

- `question`: The main question or explanation to show the user (required)
- `context`: Optional context about why the agent is asking (what was tried, what failed)
- `options`: Optional list of suggested responses for the user to choose from

When options are provided, the user can either select a numbered option or type their own custom response.

Value

A Tool object that can be used with agents

Examples

```
## Not run:
# Create the tool
ask_tool <- create_ask_user_tool()

# Use in an agent
agent <- Agent$new(
  name = "helper",
  tools = list(ask_tool, other_tools...)
)

# The agent can then use it when user input is required:
# ask_user(
#   question = "This will overwrite report.html. Should I continue?",
#   context = "The target file already exists in the project directory.",
#   options = ["Overwrite it", "Choose another path", "Cancel"]
# )
```

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

`create_capture_renderer` *Create a capturing agent-output renderer*

Description

Returns a CaptureRenderer that records agent-output events instead of displaying them. Call its `events()` method to retrieve the recorded events.

Usage

```
create_capture_renderer()
```

Value

A CaptureRenderer.

`create_chat_session` *Create a Chat Session*

Description

Factory function to create a new ChatSession object.

Usage

```
create_chat_session(
  model = NULL,
  system_prompt = NULL,
  tools = NULL,
  hooks = NULL,
  max_steps = 10,
  metadata = NULL,
  agent = NULL
)
```

Arguments

`model`	A LanguageModelV1 object or model string ID.
`system_prompt`	Optional system prompt.
`tools`	Optional list of Tool objects.
`hooks`	Optional HookHandler object.
`max_steps`	Maximum tool execution steps. Default 10.
`metadata`	Optional session metadata (list).
`agent`	Optional Agent object to initialize from.

Value

A ChatSession object.

Examples

```
if (interactive()) {
  # Create a chat session
  chat <- create_chat_session(
    model = "openai:gpt-4o",
    system_prompt = "You are a helpful R programming assistant."
  )

  # Create from an existing agent
 agent <- create_agent("MathAgent", "Does math", system_prompt = "You are a math wizard.")
  chat <- create_chat_session(model = "openai:gpt-4o", agent = agent)

  # Send messages
  response <- chat$send("How do I read a CSV file?")
  print(response$text)

  # Continue the conversation (history is maintained)
  response <- chat$send("What about Excel files?")

  # Check stats
  print(chat$stats())

  # Save session
  chat$save("my_session.rds")
}
```

---

create_coder_agent *Create a CoderAgent*

Description

Creates an agent specialized in writing and executing R code. The agent can execute R code in the session environment, making results available to other agents. Enhanced version with better safety controls and debugging support.

Usage

```
create_coder_agent(
  name = "CoderAgent",
  safe_mode = TRUE,
  timeout_ms = 30000,
  max_output_lines = 200
)
```

Arguments

name	Agent name. Default "CoderAgent".
safe_mode	If TRUE (default), restricts file system and network access.
timeout_ms	Code execution timeout in milliseconds. Default 30000.
max_output_lines	Maximum output lines to return. Default 50.

Value

An Agent object configured for R code execution.

Examples

```
if (interactive()) {
  coder <- create_coder_agent()
  session <- create_shared_session(model = "openai:gpt-4o")
  result <- coder$run(
    "Create a data frame with 3 rows and calculate the mean",
    session = session,
    model = "openai:gpt-4o"
  )
}
```

---

create_computer_tools *Create Computer Tools*

Description

Create atomic tools for computer abstraction layer. These tools provide a small set of primitives that agents can use to perform complex actions.

Usage

```
create_computer_tools(
  computer = NULL,
  working_dir = tempdir(),
  sandbox_mode = "permissive"
)
```

Arguments

computer	Computer instance (default: create new)
working_dir	Working directory. Defaults to `tempdir()`.
sandbox_mode	Sandbox mode: "strict", "permissive", or "none"

Value

List of Tool objects

`create_console_agent` *Create Console Agent*

Description

Create the default intelligent terminal agent for console_chat(). This agent can execute commands, manage files, and run R code through natural language interaction.

Usage

```
create_console_agent(
  working_dir = tempdir(),
  sandbox_mode = "permissive",
  additional_tools = NULL,
  language = "auto",
  startup_dir = working_dir,
  skills = "auto",
  profile = c("minimal", "legacy"),
  extensions = "auto"
)
```

Arguments

`working_dir`	Working directory for sandboxed tool execution. Defaults to `tempdir()`.
`sandbox_mode`	Sandbox mode: "strict", "permissive", or "none" (default: "permissive").
`additional_tools`	Optional list of additional Tool objects to include.
`language`	Language for responses: "auto", "en", or "zh" (default: "auto").
`startup_dir`	R session startup directory used for project-aware context. Defaults to `getwd()`.
`skills`	Optional skill paths, `"auto"`, or a SkillRegistry object.
`profile`	Console profile. `"minimal"` is the default Pi-like tool set; `"legacy"` restores the previous all-in-one console agent.
`extensions`	Extension loading mode. Defaults to `"auto"`.

Value

An Agent object configured for console interaction.

Examples

```
if (interactive()) {
    # Create default console agent
    agent <- create_console_agent()

    # Create with custom working directory
    agent <- create_console_agent(working_dir = "~/projects/myapp")

    # Use with console_chat
    console_chat("openai:gpt-4o", agent = agent)
}
```

---

create_console_tools *Create Console Tools*

Description

Create a set of tools optimized for console/terminal interaction. The default `"minimal"` profile exposes only `bash`, `read_file`, `write_file`, and `edit_file`. Use `profile = "legacy"` for the prior broad tool surface including R, image, Feishu, skill, and inspection tools.

Usage

```
create_console_tools(
  working_dir = tempdir(),
  sandbox_mode = "permissive",
  startup_dir = working_dir,
  profile = c("minimal", "legacy"),
  extensions = "auto"
)
```

Arguments

`working_dir`	Working directory for sandboxed tool execution. Defaults to `tempdir()`.
`sandbox_mode`	Sandbox mode: "strict", "permissive", or "none" (default: "permissive").
`startup_dir`	R session startup directory used for project-aware context. Defaults to `getwd()`.
`profile`	Console profile. `"minimal"` is the default Pi-like tool set; `"legacy"` restores the previous all-in-one console tools.
`extensions`	Extension loading mode. Defaults to `"auto"`.

Value

A list of Tool objects.

Examples

```
if (interactive()) {
    tools <- create_console_tools()
    # Use with an agent or session
    session <- create_chat_session(model = "openai:gpt-4o", tools = tools)
}
```

`create_context_management_config`

Create Context Management Configuration

Description

Build a normalized context-management configuration list.

Usage

```
create_context_management_config(
  mode = NULL,
  llm_synthesis = FALSE,
  synthesis_model = NULL,
  llm_synthesis_policy = NULL,
  context_window_override = NULL,
  max_output_tokens_override = NULL,
  project_memory_root = NULL,
  retrieval_providers = NULL,
  retrieval_provider_order = NULL,
  retrieval_provider_limits = NULL,
  retrieval_min_hits = NULL,
  retrieval_scoring_policy = NULL,
  retrieval_reranking = FALSE,
  retrieval_reranking_model = NULL,
  retrieval_reranking_policy = NULL
)
```

Arguments

`mode`	One of `"off"`, `"basic"`, or `"adaptive"`.
`llm_synthesis`	Logical; whether to enable optional LLM enrichment for working-memory synthesis.
`synthesis_model`	Optional model reference used for LLM synthesis.
`llm_synthesis_policy`	Optional named list controlling LLM synthesis behavior. Supported fields are `min_regime`, `recent_messages`, `max_items`, and `temperature`.

`context_window_override`
: Optional numeric context-window override.

`max_output_tokens_override`
: Optional numeric max-output override.

`project_memory_root`
: Optional project root used for `ProjectMemory` retrieval and learning.

`retrieval_providers`
: Optional retrieval-provider selection. Accepts either a character vector of enabled providers or a named logical/list map.

`retrieval_provider_order`
: Optional character vector controlling render and ranking priority across retrieval providers.

`retrieval_provider_limits`
: Optional named list of per-provider result limits.

`retrieval_min_hits`
: Optional named list of per-provider matching thresholds.

`retrieval_scoring_policy`
: Optional named list controlling cross-provider retrieval scoring. Supported fields are `provider_weights`, `token_match_weight`, `exact_query_bonus`, `title_match_weight`, `summary_match_weight`, `preview_match_weight`, `bigram_match_weight`, `coverage_weight`, `workflow_hint_weight`, `accessor_match_weight`, `semantic_object_bonus`, `task_signal_weight`, `execution_error_weight`, `system_metric_weight`, `runtime_signal_weight` `source_kind_bonus`, `recency_weight`, and `max_total_results`.

`retrieval_reranking`
: Logical; whether to enable optional learned reranking on top of deterministic retrieval scoring.

`retrieval_reranking_model`
: Optional model reference used for reranking.

`retrieval_reranking_policy`
: Optional named list controlling reranking. Supported fields are `min_regime`, `top_n`, and `temperature`.

Value

A normalized configuration list.

`create_context_query_tools` *Create Context Query Tools*

Description

Create ordinary `Tool` objects for searching handles, resolving handles, peeking live R objects, and launching bounded child sessions.

Usage

```
create_context_query_tools(session = NULL)
```

Arguments

session	Optional ChatSession or SharedSession captured by the tools.

Value

A list of Tool objects.

---

create_custom_provider *Create a custom provider*

Description

Creates a dynamic wrapper around existing model classes (OpenAI, Anthropic) based on user-provided configuration. The returned provider can be registered in the global ProviderRegistry.

Usage

```
create_custom_provider(
  provider_name,
  base_url,
  api_key = NULL,
  api_format = c("chat_completions", "responses", "anthropic_messages"),
  use_max_completion_tokens = FALSE,
  disable_stream_options = TRUE,
  supports_native_tools = FALSE
)
```

Arguments

provider_name	The identifier name for this custom provider (e.g. "my_custom_openai_proxy").
base_url	The base URL for the API endpoint.
api_key	The API key for authentication. If NULL, defaults to checking environmental variables.
api_format	The underlying API format to use. Supports "chat_completions" (OpenAI default), "responses" (OpenAI Responses API), and "anthropic_messages" (Anthropic Messages API).
use_max_completion_tokens	A boolean flag. If TRUE, injects the is_reasoning_model capability to ensure the model uses max_completion_tokens instead of max_tokens.

disable_stream_options
: A boolean flag. If TRUE, omit OpenAI-style stream_options from streaming requests. Useful for self-hosted OpenAI-compatible gateways that only implement the basic streaming shape.

supports_native_tools
: A boolean flag. If FALSE, do not send native OpenAI/Anthropic tool definitions or tool-result message formats. The SDK will fall back to text-embedded <tool_call>{...}</tool_call> blocks.

Value

A custom provider object with a language_model(model_id) method.

---

create_data_agent *Create a DataAgent*

Description

Creates an agent specialized in data manipulation using dplyr and tidyr. The agent can filter, transform, summarize, and reshape data frames in the session environment.

Usage

```
create_data_agent(name = "DataAgent", safe_mode = TRUE)
```

Arguments

name	Agent name. Default "DataAgent".
safe_mode	If TRUE (default), restricts operations to data manipulation only.

Value

An Agent object configured for data manipulation.

Examples

```
if (interactive()) {
  data_agent <- create_data_agent()
  session <- create_shared_session(model = "openai:gpt-4o")
  session$set_var("sales", data.frame(
    product = c("A", "B", "C"),
    revenue = c(100, 200, 150)
  ))
  result <- data_agent$run(
    "Calculate total revenue and find the top product",
    session = session,
    model = "openai:gpt-4o"
  )
}
```

create_default_semantic_adapter_registry *Create the Default Semantic Adapter Registry*

Description

Build the package default semantic adapter registry used for semantic object inspection. The default registry includes built-in generic adapters and can optionally apply extension registrars.

Usage

```
create_default_semantic_adapter_registry(
  include_workflow_hints = TRUE,
  extension_registrars = NULL
)
```

Arguments

include_workflow_hints	Logical; whether extension registrars should register workflow hints.
extension_registrars	Optional list of registrar functions. Each registrar is called with `registry` and `include_workflow_hints`.

Details

This function is safe for extension packages that want the standard `aisdk` registry baseline before registering additional adapters.

Value

A `SemanticAdapterRegistry` object.

create_embeddings *Create Embeddings*

Description

Generate embeddings for text using an embedding model.

Usage

```
create_embeddings(model, value, registry = NULL)
```

Arguments

model	Either an EmbeddingModelV1 object, or a string ID like "openai:text-embedding-3-small".
value	A character string or vector to embed.
registry	Optional ProviderRegistry to use.

Value

A list with embeddings and usage information.

Examples

```
if (interactive()) {
  model <- create_openai()$embedding_model("text-embedding-3-small")
  result <- create_embeddings(model, "Hello, world!")
  print(length(result$embeddings[[1]]))
}
```

---

create_env_agent *Create an EnvAgent*

Description

Creates an agent specialized in R environment and package management. The agent can check, install, and manage R packages with safety controls.

Usage

```
create_env_agent(
  name = "EnvAgent",
  allow_install = FALSE,
  allowed_repos = "https://cloud.r-project.org"
)
```

Arguments

name	Agent name. Default "EnvAgent".
allow_install	Allow package installation. Default FALSE.
allowed_repos	CRAN mirror URLs for installation.

Value

An Agent object configured for environment management.

Examples

```
if (interactive()) {
  env_agent <- create_env_agent(allow_install = TRUE)
  result <- env_agent$run(
    "Check if tidyverse is installed and load it",
    session = session,
    model = "openai:gpt-4o"
  )
}
```

`create_file_agent` *Create a FileAgent*

Description

Creates an agent specialized in file system operations using fs and readr. The agent can read, write, and manage files with safety guardrails.

Usage

```
create_file_agent(
  name = "FileAgent",
  allowed_dirs = ".",
  allowed_extensions = c("csv", "tsv", "txt", "json", "rds", "rda", "xlsx", "xls")
)
```

Arguments

`name`	Agent name. Default "FileAgent".
`allowed_dirs`	Character vector of allowed directories. Default current dir.
`allowed_extensions`	Character vector of allowed file extensions.

Value

An Agent object configured for file operations.

Examples

```
if (interactive()) {
  file_agent <- create_file_agent(
    allowed_dirs = c("./data", "./output"),
    allowed_extensions = c("csv", "json", "txt", "rds")
  )
  result <- file_agent$run(
    "Read the sales.csv file and store it as 'sales_data'",
    session = session,
```

```
    model = "openai:gpt-4o"
  )
}
```

`create_gemini` *Create Gemini Provider*

Description

Factory function to create a Gemini provider.

Usage

```
create_gemini(
  api_key = NULL,
  base_url = NULL,
  headers = NULL,
  name = NULL,
  timeout_seconds = NULL,
  total_timeout_seconds = NULL,
  first_byte_timeout_seconds = NULL,
  connect_timeout_seconds = NULL,
  idle_timeout_seconds = NULL
)
```

Arguments

`api_key`	Gemini API key. Defaults to GEMINI_API_KEY env var.
`base_url`	Base URL for API calls. Defaults to https://generativelanguage.googleapis.com/v1beta/models.
`headers`	Optional additional headers.
`name`	Optional provider name override.
`timeout_seconds`	Legacy alias for `total_timeout_seconds`.
`total_timeout_seconds`	Optional total request timeout in seconds for API calls.
`first_byte_timeout_seconds`	Optional time-to-first-byte timeout in seconds for API calls.
`connect_timeout_seconds`	Optional connection-establishment timeout in seconds for API calls.
`idle_timeout_seconds`	Optional stall timeout in seconds for API calls.

Value

A GeminiProvider object.

Supported Models

- **gemini-3-pro-preview**: Preview Gemini 3 Pro model for advanced reasoning, coding, multimodal understanding, and agentic workflows. (Reasoning, Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-3-pro-preview-customtools**: Gemini 3 Pro Preview endpoint tuned for agentic workflows that combine bash-like and custom tools. (Reasoning, Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-3-flash-preview**: Preview Gemini 3 Flash model for frontier-class performance at lower latency and cost. (Reasoning, Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-2.5-pro**: Stable Gemini 2.5 Pro model for complex reasoning, coding, STEM, and long-context analysis. (Reasoning, Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-2.5-flash**: Stable Gemini 2.5 Flash model for price-performance, low-latency, high-volume, and agentic tasks. (Reasoning, Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-2.5-flash-lite**: Stable Gemini 2.5 Flash-Lite model for cost-efficient high-throughput multimodal tasks. (Reasoning, Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-2.0-flash**: Deprecated Gemini 2.0 Flash model with 1M context; migrate to Gemini 3 Flash Preview or a newer model. (Vision, Tools, Audio, Structured, Search) | ctx: 1049k
- **gemini-2.0-flash-lite**: Deprecated Gemini 2.0 Flash-Lite model; migrate to Gemini 2.5 Flash-Lite or a newer model. (Vision, Tools, Audio, Structured) | ctx: 1049k
- **gemini-3-pro-image-preview**: Preview Gemini 3 Pro image generation and editing model for professional-grade visuals and accurate text rendering. (Reasoning, Vision, Image-Out, Image-Edit, Structured, Search) | ctx: 66k
- **gemini-2.5-flash-image**: Stable Gemini 2.5 Flash image generation and conversational editing model for high-volume creative workflows. (Vision, Image-Out, Image-Edit, Structured) | ctx: 66k

Examples

```
if (interactive()) {
    gemini <- create_gemini(api_key = "AIza...")
    model <- gemini$language_model("gemini-2.5-flash")
}
```

---

create_hooks *Create Hooks*

Description

Helper to create a HookHandler from a list of functions.

Usage

```
create_hooks(...)
```

Arguments

...	Named arguments matching supported hook names.

Value

A HookHandler object.

---

create_null_renderer *Create a null (no-op) agent-output renderer*

Description

Returns a Renderer that discards every event. Use it for headless or library contexts where agent output should not be displayed.

Usage

```
create_null_renderer()
```

Value

A Renderer.

---

create_openai *Create OpenAI Provider*

Description

Factory function to create an OpenAI provider.

Usage

```
create_openai(
  api_key = NULL,
  base_url = NULL,
  organization = NULL,
  project = NULL,
  headers = NULL,
  name = NULL,
  timeout_seconds = NULL,
  total_timeout_seconds = NULL,
  first_byte_timeout_seconds = NULL,
  connect_timeout_seconds = NULL,
  idle_timeout_seconds = NULL,
  disable_stream_options = FALSE,
  api_format = c("auto", "chat", "responses")
)
```

Arguments

`api_key`	OpenAI API key. Defaults to OPENAI_API_KEY env var.
`base_url`	Base URL for API calls. Defaults to https://api.openai.com/v1.
`organization`	Optional OpenAI organization ID.
`project`	Optional OpenAI project ID.
`headers`	Optional additional headers.
`name`	Optional provider name override (for compatible APIs).
`timeout_seconds`	Legacy alias for `total_timeout_seconds`.
`total_timeout_seconds`	Optional total request timeout in seconds for API calls.
`first_byte_timeout_seconds`	Optional time-to-first-byte timeout in seconds for API calls.
`connect_timeout_seconds`	Optional connection-establishment timeout in seconds for API calls.
`idle_timeout_seconds`	Optional stall timeout in seconds for API calls.
`disable_stream_options`	Disable stream_options parameter (for providers like Volcengine that don't support it).
`api_format`	Default API surface for `smart_model()` / `model()`: `"auto"` (default, picks Chat or Responses based on model), `"chat"` (always Chat Completions), or `"responses"` (always Responses API).

Value

An OpenAIProvider object.

Supported Models

- **gpt-5.5**: Latest OpenAI frontier model for complex reasoning, coding, and professional workflows. (Reasoning, Vision, Tools, Structured) | ctx: 1050k
- **gpt-5.5-pro**: Higher-compute GPT-5.5 variant for the hardest professional and reasoning tasks. (Reasoning, Vision, Tools, Structured) | ctx: 1050k
- **gpt-5.4**: Frontier model for coding and professional work with 1.05M context. (Reasoning, Vision, Tools, Structured) | ctx: 1050k
- **gpt-5.4-pro**: Higher-compute GPT-5.4 variant for more precise responses on difficult tasks. (Reasoning, Vision, Tools, Structured) | ctx: 1050k
- **gpt-5.4-mini**: Strong mini model for coding, computer use, and subagents. (Reasoning, Vision, Tools, Structured) | ctx: 400k
- **gpt-5.4-nano**: Lowest-cost GPT-5.4-class model for high-volume classification, extraction, ranking, and subagents. (Reasoning, Vision, Tools, Structured) | ctx: 400k
- **gpt-5.3-codex**: Agentic coding model optimized for Codex-like environments. (Reasoning, Vision, Tools, Structured) | ctx: 400k

- **gpt-5**: Reasoning model for coding and agentic tasks across domains. (Reasoning, Vision, Tools, Structured) | ctx: 400k
- **gpt-5-mini**: Cost-sensitive, low-latency GPT-5 model for well-defined tasks and precise prompts. (Reasoning, Vision, Tools, Structured) | ctx: 400k
- **gpt-5-nano**: Fastest and lowest-cost GPT-5 model for summarization, classification, and routing. (Reasoning, Vision, Tools, Structured) | ctx: 400k
- **gpt-4.1**: Smart non-reasoning model with strong instruction following and tool calling. (Vision, Tools, Structured) | ctx: 1048k
- **gpt-4.1-mini**: Smaller, faster GPT-4.1 model with long-context support. (Vision, Tools, Structured) | ctx: 1048k
- **gpt-4.1-nano**: Fastest and lowest-cost GPT-4.1 model for simple tasks. (Vision, Tools, Structured) | ctx: 1048k
- **gpt-4o**: Legacy multimodal GPT model with broad ecosystem support. (Vision, Tools, Structured) | ctx: 128k
- **gpt-4o-mini**: Fast, affordable GPT-4o model for focused tasks. (Vision, Tools, Structured) | ctx: 128k
- *... and 6 more models. Use* `list_models("openai")` *to see all.*

Token Limit Parameters

The SDK provides a unified `max_tokens` parameter that automatically maps to the correct API field based on the model and API type:

- **Chat API (standard models)**: `max_tokens` -> `max_tokens`
- **Chat API (o1/o3 models)**: `max_tokens` -> `max_completion_tokens`
- **Responses API**: `max_tokens` -> `max_output_tokens` (total: reasoning + answer)

For advanced users who need fine-grained control:

- `max_completion_tokens`: Explicitly set completion tokens (Chat API, o1/o3)
- `max_output_tokens`: Explicitly set total output limit (Responses API)
- `max_answer_tokens`: Limit answer only, excluding reasoning (Responses API, Volcengine-specific)

Examples

```
if (interactive()) {
  # Basic usage with Chat Completions API
  openai <- create_openai(api_key = "sk-...")
  model <- openai$language_model("gpt-4o")
  result <- generate_text(model, "Hello!")

  # Using Responses API for reasoning models
  openai <- create_openai()
  model <- openai$responses_model("o1")
  result <- generate_text(model, "Solve this math problem...")
  print(result$reasoning) # Access chain-of-thought
```

```
  # Smart model selection (auto-detects best API)
  model <- openai$smart_model("o3-mini") # Uses Responses API
  model <- openai$smart_model("gpt-4o") # Uses Chat Completions API

  # Token limits - unified interface
  # For standard models: limits generated content
  result <- model$generate(messages = msgs, max_tokens = 1000)

  # For o1/o3 models: automatically maps to max_completion_tokens
  model_o1 <- openai$language_model("o1")
  result <- model_o1$generate(messages = msgs, max_tokens = 2000)

  # For Responses API: automatically maps to max_output_tokens (total limit)
  model_resp <- openai$responses_model("o1")
  result <- model_resp$generate(messages = msgs, max_tokens = 2000)

  # Advanced: explicitly control answer-only limit (Volcengine Responses API)
  result <- model_resp$generate(messages = msgs, max_answer_tokens = 500)

  # Multi-turn conversation with Responses API
  model <- openai$responses_model("o1")
  result1 <- generate_text(model, "What is 2+2?")
  result2 <- generate_text(model, "Now multiply that by 3") # Remembers context
  model$reset() # Start fresh conversation
}
```

`create_permission_hook` *Create Permission Hook*

Description

Create a hook that enforces a permission mode for tool execution.

Usage

```
create_permission_hook(
  mode = c("implicit", "explicit", "escalate"),
  allowlist = c("search_web", "read_resource", "read_file")
)
```

Arguments

`mode`	Permission mode: • "implicit" (default): Auto-approve all tools. • "explicit": Ask for confirmation in the console for every tool. • "escalate": Ask for confirmation only for tools not in the allowlist.

allowlist — List of tool names that are auto-approved in "escalate" mode. Default includes read-only tools like "search_web", "read_file".

Value

A HookHandler object.

`create_planner_agent` *Create a PlannerAgent*

Description

Creates an agent specialized in breaking down complex tasks into steps using chain-of-thought reasoning. The planner helps decompose problems and create action plans.

Usage

```
create_planner_agent(name = "PlannerAgent")
```

Arguments

name — Agent name. Default "PlannerAgent".

Value

An Agent object configured for planning and reasoning.

Examples

```
if (interactive()) {
  planner <- create_planner_agent()
  result <- planner$run(
    "How should I approach building a machine learning model for customer churn?",
    model = "openai:gpt-4o"
  )
}
```

create_r_code_tool *Create R Code Interpreter Tool*

Description

Creates a meta-tool (`execute_r_code`) backed by a SandboxManager. This single tool replaces all individual tools for the LLM, enabling batch execution, data filtering, and local computation.

Usage

```
create_r_code_tool(sandbox)
```

Arguments

sandbox	A SandboxManager object.

Value

A Tool object named `execute_r_code`.

create_r_context_tools *Create R Context Tools*

Description

Create built-in read-only tools for live R context inspection.

Usage

```
create_r_context_tools()
```

Value

A list of Tool objects.

create_r_introspect_tools
Create R Introspection Tools

Description

Build Tool objects that let an Agent enrich diagnostic context on its own.

Usage

```
create_r_introspect_tools()
```

Value

A list of two Tool objects: `r_eval` and `r_session_state`.

Examples

```
## Not run:
tools <- create_r_introspect_tools()
agent <- create_agent(
  name = "Diagnostician",
  tools = tools,
  model = "openai:gpt-4o"
)

## End(Not run)
```

create_sandbox_system_prompt
Create Sandbox System Prompt

Description

Generates a system prompt section that instructs the LLM how to use the R code sandbox effectively.

Usage

```
create_sandbox_system_prompt(sandbox)
```

Arguments

sandbox	A SandboxManager object.

Value

A character string to append to the system prompt.

`create_schema_from_func` *Create Schema from Function*

Description

Inspects an R function and generates a z_object schema based on its arguments and default values.

Usage

```
create_schema_from_func(
  func,
  include_args = NULL,
  exclude_args = NULL,
  params = NULL,
  func_name = NULL,
  type_mode = c("infer", "any")
)
```

Arguments

`func`	The R function to inspect.
`include_args`	Optional character vector of argument names to include. If provided, only these arguments will be included in the schema.
`exclude_args`	Optional character vector of argument names to exclude.
`params`	Optional named list of parameter values to use as defaults. This allows overriding the function's default values (e.g., with values extracted from an existing plot layer).
`func_name`	Optional string of the function name to look up documentation. If not provided, attempts to infer from 'func' symbol.
`type_mode`	How to assign parameter types. "infer" (default) uses default values to infer types. "any" uses z_any() for all parameters.

Value

A z_object schema.

Examples

```
if (interactive()) {
my_func <- function(a = 1, b = "text", c = TRUE) {}
schema <- create_schema_from_func(my_func)
print(schema)

# Override defaults
schema_override <- create_schema_from_func(my_func, params = list(a = 99))
}
```

`create_semantic_adapter` *Create a Semantic Adapter*

Description

Build a semantic adapter that can describe and inspect domain-specific R objects. Adapters are registered into a semantic adapter registry and selected by `supports(obj)` predicates at runtime.

Usage

```
create_semantic_adapter(
  name,
  supports,
  capabilities = character(0),
  priority = 0,
  describe_identity = NULL,
  describe_schema = NULL,
  describe_semantics = NULL,
  peek = NULL,
  slice = NULL,
  list_accessors = NULL,
  estimate_cost = NULL,
  provenance = NULL,
  validate_action = NULL,
  render_summary = NULL,
  render_inspection = NULL
)
```

Arguments

`name`	Adapter name.
`supports`	Function that returns `TRUE` when the adapter supports `obj`.
`capabilities`	Character vector of supported capabilities.
`priority`	Numeric priority; higher values win during dispatch.
`describe_identity`	Optional function returning identity metadata.
`describe_schema`	Optional function returning schema metadata.
`describe_semantics`	Optional function returning semantic summary metadata.
`peek`	Optional function for budget-aware preview generation.
`slice`	Optional function for budget-aware slicing.
`list_accessors`	Optional function returning recommended accessors.

estimate_cost	Optional function returning compute/token/I/O estimates.
provenance	Optional function returning provenance metadata.
validate_action	Optional function for safety validation.
render_summary	Optional function returning a human-readable summary.
render_inspection	Optional function returning a human-readable inspection.

Details

This constructor is part of the public semantic adapter authoring API for extension packages.

Adapter callbacks are optional unless noted otherwise.

`supports(obj)` should return a single `TRUE` or `FALSE` value and must not error for unsupported objects.

The structured callbacks `describe_identity()`, `describe_schema()`, `describe_semantics()`, `estimate_cost()`, and `provenance()` should return named lists when supplied.

`list_accessors()` should return a character vector of recommended accessors.

`validate_action()` should return a named list with fields such as `status`, `reason`, `category`, and `expensive`.

`render_summary()` and `render_inspection()` should return single character strings suitable for user-facing output.

Value

A `SemanticAdapter` object.

---

create_semantic_adapter_registry

Create a Semantic Adapter Registry

Description

Create a registry for semantic adapters. The registry resolves the highest priority adapter whose `supports(obj)` predicate matches an object.

Usage

```
create_semantic_adapter_registry(adapters = list())
```

Arguments

adapters	Optional list of adapters to register on creation.

Details

This constructor is part of the public semantic adapter authoring API for extension packages.

The returned registry exposes `register()`, `unregister()`, `get_adapter()`, `list_adapters()`, `register_workflow_hint()`, `list_workflow_hints()`, `resolve_workflow_hint()`, and `resolve()` methods.

Value

A `SemanticAdapterRegistry` object.

`create_session` *Create Session (Compatibility Wrapper)*

Description

Creates a session using either the new SharedSession or legacy ChatSession based on feature flags. This provides a migration path for existing code.

Usage

```
create_session(
  model = NULL,
  system_prompt = NULL,
  tools = NULL,
  hooks = NULL,
  max_steps = 10,
  ...
)
```

Arguments

`model`	A LanguageModelV1 object or model string ID.
`system_prompt`	Optional system prompt.
`tools`	Optional list of Tool objects.
`hooks`	Optional HookHandler object.
`max_steps`	Maximum tool execution steps. Default 10.
`...`	Additional arguments passed to session constructor.

Value

A SharedSession or ChatSession object.

Examples

```
if (interactive()) {
# Automatically uses SharedSession if feature enabled
session <- create_session(model = "openai:gpt-4o")

# Force legacy session
sdk_set_feature("use_shared_session", FALSE)
session <- create_session(model = "openai:gpt-4o")
}
```

---

create_shared_session *Create a Shared Session*

Description

Factory function to create a new SharedSession object.

Usage

```
create_shared_session(
  model = NULL,
  system_prompt = NULL,
  tools = NULL,
  hooks = NULL,
  max_steps = 10,
  sandbox_mode = "strict",
  trace_enabled = TRUE
)
```

Arguments

model	A LanguageModelV1 object or model string ID.
system_prompt	Optional system prompt.
tools	Optional list of Tool objects.
hooks	Optional HookHandler object.
max_steps	Maximum tool execution steps. Default 10.
sandbox_mode	Sandbox mode: "strict", "permissive", or "none". Default "strict".
trace_enabled	Enable execution tracing. Default TRUE.

Value

A SharedSession object.

Examples

```
if (interactive()) {
# Create a shared session for multi-agent use
session <- create_shared_session(
  model = "openai:gpt-4o",
  sandbox_mode = "strict",
  trace_enabled = TRUE
)

# Execute code safely
result <- session$execute_code("x <- 1:10; mean(x)")

# Check trace
print(session$trace_summary())
}
```

`create_skill_architect_agent` *Create a SkillArchitect Agent*

Description

Creates an advanced agent specialized in creating, testing, and refining new skills. It follows a rigorous "Ingest -> Design -> Implement -> Verify" workflow.

Usage

```
create_skill_architect_agent(
  name = "SkillArchitect",
  registry = NULL,
  model = NULL
)
```

Arguments

`name`	Agent name. Default "SkillArchitect".
`registry`	Optional SkillRegistry object (defaults to creating one from inst/skills).
`model`	The model object to use for verification (spawning a tester agent).

Value

An Agent object configured for skill architecture.

create_skill_registry *Create a Skill Registry*

Description

Convenience function to create and populate a SkillRegistry.

Usage

```
create_skill_registry(path, recursive = FALSE)
```

Arguments

path	Path to scan for skills.
recursive	Whether to scan subdirectories. Default FALSE.

Value

A populated SkillRegistry object.

Examples

```
if (interactive()) {
# Scan a skills directory
registry <- create_skill_registry(".aimd/skills")

# List available skills
registry$list_skills()

# Get a specific skill
skill <- registry$get_skill("seurat_analysis")
}
```

create_skill_tools *Create Skill Tools*

Description

Create the built-in tools for interacting with skills.

Usage

```
create_skill_tools(registry)
```

Arguments

registry	A SkillRegistry object.

Value

A list of Tool objects.

---

create_standard_registry
Create Standard Agent Registry

Description

Creates an AgentRegistry pre-populated with standard library agents based on feature flags.

Usage

```
create_standard_registry(
  include_data = TRUE,
  include_file = TRUE,
  include_env = TRUE,
  include_coder = TRUE,
  include_visualizer = TRUE,
  include_planner = TRUE,
  file_allowed_dirs = ".",
  env_allow_install = FALSE
)
```

Arguments

include_data	Include DataAgent. Default TRUE.
include_file	Include FileAgent. Default TRUE.
include_env	Include EnvAgent. Default TRUE.
include_coder	Include CoderAgent. Default TRUE.
include_visualizer	Include VisualizerAgent. Default TRUE.
include_planner	Include PlannerAgent. Default TRUE.
file_allowed_dirs	Allowed directories for FileAgent.
env_allow_install	Allow package installation for EnvAgent.

Value

An AgentRegistry object.

Examples

```
if (interactive()) {
# Create registry with all standard agents
registry <- create_standard_registry()

# Create registry with only data and visualization agents
registry <- create_standard_registry(
  include_file = FALSE,
  include_env = FALSE,
  include_planner = FALSE
)
}
```

`create_telemetry` *Create Telemetry*

Description

Create Telemetry

Usage

```
create_telemetry(trace_id = NULL, emit = TRUE)
```

Arguments

`trace_id`	Optional trace ID.
`emit`	Logical; when TRUE, events are printed as JSON lines.

Value

A Telemetry object.

`create_visualizer_agent`
Create a VisualizerAgent

Description

Creates an agent specialized in creating data visualizations using ggplot2. Enhanced version with plot type recommendations, theme support, and automatic data inspection.

Usage

```
create_visualizer_agent(
  name = "VisualizerAgent",
  output_dir = NULL,
  default_theme = "theme_minimal",
  default_width = 8,
  default_height = 6
)
```

Arguments

name	Agent name. Default "VisualizerAgent".
output_dir	Optional directory to save plots. If NULL, plots are stored in the session environment.
default_theme	Default ggplot2 theme. Default "theme_minimal".
default_width	Default plot width in inches. Default 8.
default_height	Default plot height in inches. Default 6.

Value

An Agent object configured for data visualization.

Examples

```
if (interactive()) {
  visualizer <- create_visualizer_agent()
  session <- create_shared_session(model = "openai:gpt-4o")
  session$set_var("df", data.frame(x = 1:10, y = (1:10)^2))
  result <- visualizer$run(
    "Create a scatter plot of df showing the relationship between x and y",
    session = session,
    model = "openai:gpt-4o"
  )
}
```

---

create_z_ggtree *Create Schema for ggtree Function*

Description

Specialized wrapper around create_schema_from_func for ggtree/ggplot2 functions. Handles common mapping and data arguments specifically.

Usage

```
create_z_ggtree(func, layer = NULL)
```

Arguments

func	The R function (e.g., geom_tiplab).
layer	Optional ggplot2 Layer object. If provided, its parameters (aes_params and geom_params) will be used to override the schema defaults. This is useful for creating "Edit Mode" forms for existing plot layers.

Value

A z_object schema.

---

default_skill_roots *Default Skill Roots*

Description

Return the standard skill search roots in increasing priority order: bundled package skills, installed user skills, project skills, then explicitly configured roots.

Usage

```
default_skill_roots(project_dir = getwd(), include_missing = FALSE)
```

Arguments

project_dir	Project directory used for project-local skill roots.
include_missing	If TRUE, include roots even when they do not exist.

Value

Character vector of skill root directories.

---

edit_image *Edit Images*

Description

Edit images using an image editing model.

Usage

```
edit_image(
  model,
  image,
  prompt = NULL,
  mask = NULL,
  output_dir = tempdir(),
  registry = NULL,
  ...
)
```

Arguments

model	An ImageModelV1 object or provider:model string.
image	Image source or provider-specific image input.
prompt	Editing instructions.
mask	Optional mask image.
output_dir	Directory where edited images should be written. Defaults to tempdir().
registry	Optional provider registry.
...	Additional arguments passed to the model implementation.

Value

A GenerateImageResult.

EmbeddingModelV1 *Embedding Model V1 (Abstract Base Class)*

Description

Abstract interface for embedding models.

Public fields

specification_version The version of this specification.

provider The provider identifier.

model_id The model identifier.

Methods

Public methods:

- EmbeddingModelV1$new()
- EmbeddingModelV1$do_embed()
- EmbeddingModelV1$clone()

Method `new()`**:** Initialize the embedding model.

Usage:

```
EmbeddingModelV1$new(provider, model_id)
```

Arguments:

`provider` Provider name.

`model_id` Model ID.

Method `do_embed()`**:** Generate embeddings for a value. Abstract method.

Usage:

```
EmbeddingModelV1$do_embed(value)
```

Arguments:

`value` A character string or vector to embed.

Returns: A provider-specific embedding result.

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

Usage:

```
EmbeddingModelV1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`enable_api_tests` *Check if API tests should be enabled*

Description

Check if API tests should be enabled

Usage

```
enable_api_tests()
```

Value

TRUE if API tests are enabled and keys are available, FALSE otherwise

`execute_tool_calls` *Execute Tool Calls*

Description

Execute a list of tool calls returned by an LLM. This function safely executes each tool, handling errors gracefully and returning a standardized result format.

Implements multi-layer defense strategy:

1. Tool name repair (case fixing, snake_case conversion, fuzzy matching)
2. Invalid tool routing for graceful degradation
3. Argument parsing with JSON repair
4. Error capture and structured error responses

Usage

```
execute_tool_calls(
  tool_calls,
  tools,
  hooks = NULL,
  envir = NULL,
  repair_enabled = TRUE
)
```

Arguments

`tool_calls`	A list of tool call objects, each with id, name, and arguments.
`tools`	A list of Tool objects to search for matching tools.
`hooks`	Optional HookHandler object.
`envir`	Optional environment in which to execute tools. When provided, tool functions can access and modify variables in this environment, enabling cross-agent data sharing through a shared session environment.
`repair_enabled`	Whether to attempt tool call repair (default TRUE).

Value

A list of execution results, each containing:

- id: The tool call ID
- name: The tool name
- result: The execution result (or error message)
- is_error: TRUE if an error occurred during execution

`expect_llm_pass` *Expect LLM Pass*

Description

Custom testthat expectation that evaluates whether an LLM response meets specified criteria. Uses an LLM judge to assess the response.

Usage

```
expect_llm_pass(response, criteria, model = NULL, threshold = 0.7, info = NULL)
```

Arguments

`response`	The LLM response to evaluate (text or GenerateResult object).
`criteria`	Character string describing what constitutes a passing response.
`model`	Model to use for judging (default: same as response or gpt-4o).
`threshold`	Minimum score (0-1) to pass (default: 0.7).
`info`	Additional information to include in failure message.

Value

Invisibly returns the evaluation result.

Examples

```
if (interactive()) {
  test_that("agent answers math questions correctly", {
    result <- generate_text(
      model = "openai:gpt-4o",
      prompt = "What is 2 + 2?"
    )
    expect_llm_pass(result, "The response should contain the number 4")
  })
}
```

expect_no_hallucination *Expect No Hallucination*

Description

Test that an LLM response does not contain hallucinated information when compared against ground truth.

Usage

```
expect_no_hallucination(
  response,
  ground_truth,
  model = NULL,
  tolerance = 0.1,
  info = NULL
)
```

Arguments

response	The LLM response to check.
ground_truth	The factual information to check against.
model	Model to use for checking.
tolerance	Allowed deviation (0 = strict, 1 = lenient).
info	Additional information for failure message.

expect_tool_selection *Expect Tool Selection*

Description

Test that an agent selects the correct tool(s) for a given task.

Usage

```
expect_tool_selection(result, expected_tools, exact = FALSE, info = NULL)
```

Arguments

result	A GenerateResult object from generate_text with tools.
expected_tools	Character vector of expected tool names.
exact	If TRUE, require exactly these tools (no more, no less).
info	Additional information for failure message.

extension_runtime *Console Extension Runtime*

Description

Lightweight extension registry for console slash commands and optional tools.

extract_from_image *Extract Structured Data From Image*

Description

Extract schema-constrained structured data from an image using a multimodal language model.

Usage

```
extract_from_image(
  model,
  image,
  schema,
  prompt = "Extract the requested structured information from this image.",
  system = NULL,
  registry = NULL,
  ...
)
```

Arguments

model	A LanguageModelV1 object or provider:model string.
image	Image source accepted by input_image().
schema	A z_schema object used as response_format.
prompt	Prompt describing the extraction task.
system	Optional system prompt.
registry	Optional provider registry.
...	Additional arguments passed to generate_text().

Value

A GenerateResult.

fetch_api_models *Fetch available models from API provider*

Description

Fetch available models from API provider

Usage

```
fetch_api_models(provider, api_key = NULL, base_url = NULL)
```

Arguments

provider	Provider name ("openai", "nvidia", "anthropic", etc.)
api_key	API key
base_url	Base URL

Value

A data frame with 'id' column and capability flag columns

GeminiProvider *Gemini Provider Class*

Description

Provider class for Google Gemini.

Public fields

`specification_version` Provider spec version.

Methods

Public methods:

- `GeminiProvider$new()`
- `GeminiProvider$language_model()`
- `GeminiProvider$image_model()`
- `GeminiProvider$clone()`

Method `new()`**:** Initialize the Gemini provider.

Usage:

```
GeminiProvider$new(
  api_key = NULL,
  base_url = NULL,
  headers = NULL,
  name = NULL,
  timeout_seconds = NULL,
  total_timeout_seconds = NULL,
  first_byte_timeout_seconds = NULL,
  connect_timeout_seconds = NULL,
  idle_timeout_seconds = NULL
)
```

Arguments:

`api_key` Gemini API key. Defaults to GEMINI_API_KEY env var.

`base_url` Base URL for API calls. Defaults to https://generativelanguage.googleapis.com/v1beta/models.

`headers` Optional additional headers.

`name` Optional provider name override.

`timeout_seconds` Legacy alias for `total_timeout_seconds`.

`total_timeout_seconds` Optional total request timeout in seconds for API calls.

`first_byte_timeout_seconds` Optional time-to-first-byte timeout in seconds for API calls.

`connect_timeout_seconds` Optional connection-establishment timeout in seconds for API calls.

`idle_timeout_seconds` Optional stall timeout in seconds for API calls.

Method `language_model()`**:** Create a language model.

Usage:

```
GeminiProvider$language_model(
  model_id = Sys.getenv("GEMINI_MODEL", "gemini-2.5-flash")
)
```

Arguments:

`model_id` The model ID (e.g., "gemini-1.5-pro", "gemini-1.5-flash", "gemini-2.0-flash").

Returns: A GeminiLanguageModel object.

Method `image_model()`**:** Create an image model.

Usage:

```
GeminiProvider$image_model(
  model_id = Sys.getenv("GEMINI_IMAGE_MODEL", "gemini-2.5-flash-image")
)
```

Arguments:

`model_id` The model ID for image generation.

Returns: A GeminiImageModel object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

generate_image *Generate Images*

Description

Generate images using an image generation model.

Usage

```
generate_image(model, prompt, output_dir = tempdir(), registry = NULL, ...)
```

Arguments

model	An ImageModelV1 object or provider:model string.
prompt	Prompt describing the desired image.
output_dir	Directory where generated images should be written. Defaults to tempdir().
registry	Optional provider registry.
...	Additional arguments passed to the model implementation.

Value

A GenerateImageResult.

generate_text *Generate Text*

Description

Generate text using a language model. This is the primary high-level function for non-streaming text generation.

When tools are provided, the function automatically executes tool calls and feeds results back to the LLM in a task-state driven runtime. max_steps controls one execution window; the runtime may open another window or finalize from tool observations instead of silently stopping at the boundary.

Usage

```
generate_text(
  model = NULL,
  prompt,
  system = NULL,
  temperature = 0.7,
  max_tokens = NULL,
  tools = NULL,
  max_steps = 1,
```

```
  max_tool_result_errors = 2,
  require_post_tool_protocol = FALSE,
  sandbox = FALSE,
  skills = NULL,
  session = NULL,
  hooks = NULL,
  registry = NULL,
  ...
)
```

Arguments

model	Either a LanguageModelV1 object, or a string ID like "openai:gpt-4o".
prompt	A character string prompt, or a list of messages.
system	Optional system prompt.
temperature	Sampling temperature (0-2). Default 0.7.
max_tokens	Maximum tokens to generate.
tools	Optional list of Tool objects for function calling.
max_steps	Number of model/tool steps in one execution window. Default 1. The runtime treats this as a budget checkpoint, not as a hard task stop.
max_tool_result_errors	Historical compatibility option. Tool result errors are recorded as task observations; runtime policy decides whether to continue, finalize, ask the user, or block.
require_post_tool_protocol	Logical. If TRUE, after any tool results are returned the model must either make another tool call or wrap its final answer in a `<final_answer>...</final_answer>` block. This is enabled automatically for text-based tool fallback.
sandbox	Logical. If TRUE, enables R-native programmatic sandbox mode. All tools are bound into an isolated R environment and replaced by a single `execute_r_code` meta-tool. The LLM writes R code to batch-invoke tools, filter data with dplyr/purrr, and return only summary results, dramatically reducing token usage and latency. Default FALSE.
skills	Optional path to skills directory, or a SkillRegistry object. When provided, skill tools are auto-injected and skill summaries are added to the system prompt.
session	Optional ChatSession object. When provided, tool executions run in the session's environment, enabling cross-agent data sharing.
hooks	Optional HookHandler object for intercepting events.
registry	Optional ProviderRegistry to use (defaults to global registry).
...	Additional arguments passed to the model.

Value

A GenerateResult object with text and optionally tool_calls. When max_steps > 1 and tools are used, the result includes:

- steps: Number of steps taken
- all_tool_calls: List of all tool calls made across all steps

Examples

```
if (interactive()) {
  # Using hooks
  my_hooks <- create_hooks(
    on_generation_start = function(model, prompt, tools) message("Starting..."),
    on_tool_start = function(tool, args) message("Calling tool ", tool$name)
  )
  result <- generate_text(model, "...", hooks = my_hooks)
}
```

GenerateImageResult *Generate Image Result*

Description

Result object returned by image generation and editing models.

Public fields

`images` Generated image artifacts.

`text` Optional textual companion output.

`usage` Usage information from the provider.

`finish_reason` Reason the model stopped generating.

`warnings` Any warnings from the model.

`raw_response` The raw response from the provider API.

Methods

Public methods:

- `GenerateImageResult$new()`
- `GenerateImageResult$print()`
- `GenerateImageResult$clone()`

Method `new()`**:** Initialize a GenerateImageResult object.

Usage:

```
GenerateImageResult$new(
  images = NULL,
  text = NULL,
  usage = NULL,
  finish_reason = NULL,
  warnings = NULL,
  raw_response = NULL
)
```

Arguments:

`images` Generated image artifacts.

`text` Optional text output.

`usage` Usage information.

`finish_reason` Finish reason.

`warnings` Warnings.

`raw_response` Raw provider response.

Method `print()`**:** Print method for GenerateImageResult.

Usage:

```
GenerateImageResult$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`GenerateResult` *Generate Result*

Description

Result object returned by model generation.

Details

This class uses `lock_objects = FALSE` to allow dynamic field addition. This enables the ReAct loop and other components to attach additional metadata (like `steps`, `all_tool_calls`) without modifying the class.

For models that support reasoning/thinking (like OpenAI o1/o3, DeepSeek, Claude with extended thinking), the `reasoning` field contains the model's chain-of-thought content.

For Responses API models, `response_id` contains the server-side response ID which can be used for multi-turn conversations without sending full history.

Public fields

`text` The generated text content.

`usage` Token usage information.

`finish_reason` Reason the model stopped generating.

`warnings` Any warnings from the model.

`raw_response` The raw response from the provider API.

`tool_calls` List of tool calls requested by the model.

`steps` Number of ReAct loop steps taken.

`all_tool_calls` Accumulated list of all tool calls across steps.

`reasoning` Optional chain-of-thought or reasoning content.

`response_id` Server-side response ID for Responses-style APIs.

Methods

Public methods:

- `GenerateResult$new()`
- `GenerateResult$print()`
- `GenerateResult$clone()`

Method `new()`**:** Initialize a GenerateResult object.

Usage:

```
GenerateResult$new(
  text = NULL,
  usage = NULL,
  finish_reason = NULL,
  warnings = NULL,
  raw_response = NULL,
  tool_calls = NULL,
  steps = NULL,
  all_tool_calls = NULL,
  reasoning = NULL,
  response_id = NULL
)
```

Arguments:

`text` Generated text.

`usage` Token usage.

`finish_reason` Reason for stopping.

`warnings` Warnings.

`raw_response` Raw API response.

`tool_calls` Tool calls requested by the model.

`steps` Number of ReAct steps taken.

`all_tool_calls` All tool calls across steps.

`reasoning` Chain-of-thought content.

`response_id` Server-side response ID for Responses-style APIs.

Method `print()`**:** Print method for GenerateResult.

Usage:

```
GenerateResult$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`get_anthropic_base_url` *Get Anthropic base URL from environment*

Description

Get Anthropic base URL from environment

Usage

```
get_anthropic_base_url()
```

Value

Base URL for Anthropic API (default: official)

`get_anthropic_model` *Get Anthropic model name from environment*

Description

Get Anthropic model name from environment

Usage

```
get_anthropic_model()
```

Value

Model name (default: claude-sonnet-4-20250514)

get_anthropic_model_id *Get Anthropic model ID from environment*

Description

Get Anthropic model ID from environment

Usage

```
get_anthropic_model_id()
```

Value

Model ID (default: anthropic:claude-sonnet-4-20250514)

get_capability_model *Get Capability Model*

Description

Return the configured model for a capability route.

Usage

```
get_capability_model(capability, default = NULL)
```

Arguments

capability	Capability route name.
default	Value returned when no route is configured.

Value

A model ID string, model object, or `default`.

get_context_management_config *Get Context Management Configuration*

Description

Resolve the active context-management configuration for a session.

Usage

```
get_context_management_config(session)
```

Arguments

session	A ChatSession or SharedSession.

Value

A normalized configuration list.

get_default_registry *Get Default Registry*

Description

Returns the global default provider registry, creating it if necessary.

Usage

```
get_default_registry()
```

Value

A ProviderRegistry object.

get_memory *Get or Create Global Memory*

Description

Get the global project memory instance, creating it if necessary.

Usage

```
get_memory()
```

Value

A ProjectMemory object.

get_model *Get Default Model*

Description

Returns the current package-wide default language model. This is used by high-level helpers when `model = NULL`. If no explicit default has been set, `get_model()` falls back to `getOption("aisdk.default_model")`, then to `default_model` in `aisdk.yaml`, and then to `"openai:gpt-4o"`.

Usage

```
get_model(default = "openai:gpt-4o")
```

Arguments

`default`	Fallback model identifier when no explicit default has been set.

Value

A model identifier string or a `LanguageModelV1` object.

Examples

```
get_model()
```

get_model_info *Get Full Model Info*

Description

Returns the full metadata for a single model as a list. Useful for framework internals to auto-configure parameters (e.g., max_tokens, context_window).

Usage

```
get_model_info(provider, model_id)
```

Arguments

provider	The name of the provider.
model_id	The model ID string.

Value

A list containing all available metadata for the model, or NULL if not found.

get_model_options *Get Default Model Runtime Options*

Description

Returns runtime options configured with `set_model()`, including context overrides and default generation options such as thinking settings.

Usage

```
get_model_options()
```

Value

A list with optional `context_window`, `max_output_tokens`, and `call_options` entries.

get_openai_base_url *Get OpenAI Base URL from environment*

Description

Get OpenAI Base URL from environment

Usage

```
get_openai_base_url()
```

Value

Base URL string

get_openai_embedding_model *Get OpenAI Embedding Model from environment*

Description

Get OpenAI Embedding Model from environment

Usage

```
get_openai_embedding_model()
```

Value

Model name string

get_openai_model *Get OpenAI Model from environment*

Description

Get OpenAI Model from environment

Usage

```
get_openai_model()
```

Value

Model name string

get_openai_model_id *Get OpenAI Model ID from environment*

Description

Get OpenAI Model ID from environment

Usage

```
get_openai_model_id()
```

Value

Model ID string

get_or_create_semantic_adapter_registry *Get or Create a Semantic Adapter Registry*

Description

Resolve the semantic adapter registry from an environment, or create and cache a default registry when one is not already present.

Usage

```
get_or_create_semantic_adapter_registry(envir = NULL)
```

Arguments

envir	Optional environment that stores `.semantic_adapter_registry`. When `NULL`, a new default registry is created and returned without caching.

Details

This helper is safe for extension packages that need to share the active semantic registry through a session or custom environment.

Value

A `SemanticAdapterRegistry` object.

get_r_context *Get R Context*

Description

Generates a text summary of R objects to be used as context for the LLM.

Usage

```
get_r_context(vars, envir = parent.frame())
```

Arguments

vars	Character vector of variable names to include.
envir	The environment to look for variables in. Default is parent.frame().

Value

A single string containing the summaries of the requested variables.

Examples

```
if (interactive()) {
df <- data.frame(x = 1:10, y = rnorm(10))
context <- get_r_context("df")
cat(context)
}
```

get_r_documentation *Get R Documentation*

Description

Resolve an Rd topic and return a selected section.

Usage

```
get_r_documentation(
  name,
  package = NULL,
 section = c("summary", "usage", "arguments", "details", "value", "examples", "full"),
  session = NULL,
  envir = NULL,
  max_chars = 4000L
)
```

Arguments

name	Topic or function name.
package	Optional package/namespace name.
section	One of "summary", "usage", "arguments", "details", "value", "examples", or "full".
session	Optional ChatSession or SharedSession.
envir	Optional environment. Ignored when session is provided.
max_chars	Maximum characters to return.

Value

A character string with the requested documentation.

---

get_r_source *Get R Source*

Description

Resolve function source text for a closure or method when available.

Usage

```
get_r_source(
  name,
  package = NULL,
  method = NULL,
  session = NULL,
  envir = NULL,
  max_lines = 120L
)
```

Arguments

name	Function or generic name.
package	Optional package/namespace name.
method	Optional S3 method class name.
session	Optional ChatSession or SharedSession.
envir	Optional environment. Ignored when session is provided.
max_lines	Maximum lines to return.

Value

A character string source preview or structured fallback message.

has_api_key *Check if specific provider key is available*

Description

Check if specific provider key is available

Usage

```
has_api_key(provider)
```

Arguments

provider	Provider name ("openai" or "anthropic")

Value

TRUE if key is available and valid

HookHandler *Hook Handler*

Description

R6 class to manage and execute hooks.

Public fields

`hooks` List of hook functions.

Methods

Public methods:

- `HookHandler$new()`
- `HookHandler$trigger_generation_start()`
- `HookHandler$trigger_generation_end()`
- `HookHandler$trigger_tool_start()`
- `HookHandler$trigger_tool_end()`
- `HookHandler$clone()`

Method `new()`**:** Initialize HookHandler

Usage:

```
HookHandler$new(hooks_list = list())
```

Arguments:

`hooks_list` A list of hook functions. Supported hooks:

- on_generation_start(model, prompt, tools)
- on_generation_end(result)
- on_tool_start(tool, args)
- on_tool_end(tool, result)
- on_tool_approval(tool, args) - Return TRUE to approve, FALSE to deny.

Method `trigger_generation_start()`**:** Trigger on_generation_start

Usage:

```
HookHandler$trigger_generation_start(model, prompt, tools)
```

Arguments:

`model` The language model object.

`prompt` The prompt being sent.

`tools` The list of tools provided.

Method `trigger_generation_end()`**:** Trigger on_generation_end

Usage:

```
HookHandler$trigger_generation_end(result)
```

Arguments:

`result` The generation result object.

Method `trigger_tool_start()`**:** Trigger on_tool_start

Usage:

```
HookHandler$trigger_tool_start(tool, args)
```

Arguments:

`tool` The tool object.

`args` The arguments for the tool.

Method `trigger_tool_end()`**:** Trigger on_tool_end

Usage:

```
HookHandler$trigger_tool_end(
  tool,
  result,
  success = TRUE,
  error = NULL,
  args = NULL
)
```

Arguments:

`tool` The tool object.

`result` The result from the tool execution.

`success` Logical indicating whether execution succeeded.

`error` Optional error message when execution failed.

`args` Optional tool arguments for downstream telemetry.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`hooks` *Hooks System*

Description

A system for intercepting and monitoring AI SDK events. Allows implementation of "Human-in-the-loop", logging, and validation.

`hypothesis_fix_verify` *Hypothesis-Fix-Verify Loop*

Description

Advanced self-healing execution that generates hypotheses about errors, attempts fixes, and verifies the results.

Usage

```
hypothesis_fix_verify(
  code,
  model = NULL,
  test_fn = NULL,
  max_iterations = 5,
  verbose = TRUE
)
```

Arguments

`code`	Character string of R code to execute.
`model`	LLM model for analysis.
`test_fn`	Optional function to verify the result is correct.
`max_iterations`	Maximum fix iterations.
`verbose`	Print progress.

Value

List with result, fix history, and verification status.

image_api *Image APIs*

Description

High-level helpers for image analysis and image generation workflows.

ImageModelV1 *Image Model V1 (Abstract Base Class)*

Description

Abstract interface for image generation and editing models.

Public fields

`specification_version` The version of this specification.

`provider` The provider identifier.

`model_id` The model identifier.

`capabilities` Model capability flags.

Methods

Public methods:

- `ImageModelV1$new()`
- `ImageModelV1$has_capability()`
- `ImageModelV1$generate_image()`
- `ImageModelV1$edit_image()`
- `ImageModelV1$do_generate_image()`
- `ImageModelV1$do_edit_image()`
- `ImageModelV1$do_stream_image()`
- `ImageModelV1$stream_image()`
- `ImageModelV1$clone()`

Method `new()`**:** Initialize the image model.

Usage:

```
ImageModelV1$new(provider, model_id, capabilities = list())
```

Arguments:

`provider` Provider name.

`model_id` Model ID.

`capabilities` Optional list of capability flags.

Method `has_capability()`**:** Check if the image model has a specific capability.

Usage:

```
ImageModelV1$has_capability(cap)
```

Arguments:

`cap` Capability name.

Returns: Logical.

Method `generate_image()`**:** Public image generation method.

Usage:

```
ImageModelV1$generate_image(...)
```

Arguments:

`...` Call options passed to `do_generate_image()`.

Returns: A GenerateImageResult object.

Method `edit_image()`**:** Public image editing method.

Usage:

```
ImageModelV1$edit_image(...)
```

Arguments:

`...` Call options passed to `do_edit_image()`.

Returns: A GenerateImageResult object.

Method `do_generate_image()`**:** Generate images. Abstract method.

Usage:

```
ImageModelV1$do_generate_image(params)
```

Arguments:

`params` A list of call options.

Returns: A GenerateImageResult object.

Method `do_edit_image()`**:** Edit images. Abstract method.

Usage:

```
ImageModelV1$do_edit_image(params)
```

Arguments:

`params` A list of call options.

Returns: A GenerateImageResult object.

Method `do_stream_image()`**:** Stream image generation with partial-image previews. Optional — subclasses that support it override. Default raises an informative error so `stream_image()` callers can fall back to `generate_image()` on providers without streaming support.

Usage:

```
ImageModelV1$do_stream_image(params, callback)
```

Arguments:

`params` A list of call options.

`callback` A function called with each event list.

Returns: A GenerateImageResult object with the final image.

Method `stream_image()`**:** Public image streaming method.

Usage:

```
ImageModelV1$stream_image(callback, ...)
```

Arguments:

`callback` A function receiving each event list.

`...` Call options passed to `do_stream_image()`.

Returns: A GenerateImageResult object.

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

Usage:

```
ImageModelV1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`input_image` *Create Input Image Block*

Description

Create a provider-neutral image block for multimodal message content. Supports local files, remote URLs, and data URIs.

Usage

```
input_image(x, media_type = "auto", detail = "auto")
```

Arguments

`x`	Image source as a local file path, remote URL, or data URI.
`media_type`	MIME type of the image. If `NULL` or `"auto"`, attempts to infer from the source.
`detail`	Optional image detail hint for providers that support it.

Value

A list representing an input image block.

input_text *Create Input Text Block*

Description

Create a provider-neutral text block for multimodal message content.

Usage

```
input_text(text)
```

Arguments

text	Text content.

Value

A list representing an input text block.

inspect_r_function *Inspect an R Function*

Description

Inspect function signature and provenance without dumping full source by default.

Usage

```
inspect_r_function(
  name,
  package = NULL,
  detail = c("summary", "full"),
  include_methods = FALSE,
  session = NULL,
  envir = NULL
)
```

Arguments

name	Function name.
package	Optional package/namespace name.
detail	One of "summary" or "full".
include_methods	Logical; whether to include S3/S4 method previews.
session	Optional ChatSession or SharedSession.
envir	Optional environment. Ignored when session is provided.

Value

A character string inspection.

`inspect_r_object` *Inspect a Live R Object*

Description

Inspect a symbol resolved from the live session environment.

Usage

```
inspect_r_object(
  name,
  detail = c("summary", "full", "structured"),
  session = NULL,
  envir = NULL,
  head_rows = 6L,
  scope = c("all", "session", "workspace")
)
```

Arguments

`name`	Symbol-like object name.
`detail`	One of `"summary"`, `"full"`, or `"structured"`.
`session`	Optional `ChatSession` or `SharedSession`.
`envir`	Optional environment. Ignored when `session` is provided.
`head_rows`	Maximum preview rows for inspection renderers.
`scope`	Where to resolve the object. Defaults to `"all"` so console inspection can see session objects and `.GlobalEnv` read-only.

Value

A character string for `"summary"`/`"full"` or a structured list for `"structured"`.

is_semantic_class *Check Whether an Object Belongs to a Semantic Class*

Description

Test class membership using S3 `inherits()` and S4 `methods::is()` fallback. This helper is intended for robust adapter `supports()` predicates.

Usage

```
is_semantic_class(obj, class_name)
```

Arguments

`obj`	Object to test.
`class_name`	Class name to check.

Value

`TRUE` if `obj` matches `class_name`; otherwise `FALSE`.

LanguageModelV1 *Language Model V1 (Abstract Base Class)*

Description

Abstract interface for language models. All LLM providers must implement this class. Uses `do_` prefix for internal methods to prevent direct usage by end-users.

Public fields

`specification_version` The version of this specification.

`provider` The provider identifier (e.g., "openai").

`model_id` The model identifier (e.g., "gpt-4o").

`capabilities` Model capability flags.

Methods

Public methods:

- `LanguageModelV1$new()`
- `LanguageModelV1$has_capability()`
- `LanguageModelV1$generate()`
- `LanguageModelV1$stream()`
- `LanguageModelV1$do_generate()`

- LanguageModelV1$do_stream()
- LanguageModelV1$format_tool_result()
- LanguageModelV1$get_history_format()
- LanguageModelV1$clone()

Method `new()`**:** Initialize the model with provider and model ID.

Usage:

```
LanguageModelV1$new(provider, model_id, capabilities = list())
```

Arguments:

`provider` Provider name.

`model_id` Model ID.

`capabilities` Optional list of capability flags.

Method `has_capability()`**:** Check if the model has a specific capability.

Usage:

```
LanguageModelV1$has_capability(cap)
```

Arguments:

`cap` Capability name.

Returns: Logical.

Method `generate()`**:** Public generation method.

Usage:

```
LanguageModelV1$generate(...)
```

Arguments:

`...` Call options passed to `do_generate()`.

Returns: A GenerateResult object.

Method `stream()`**:** Public streaming method.

Usage:

```
LanguageModelV1$stream(callback, ...)
```

Arguments:

`callback` Callback invoked with `(text, done)`.

`...` Call options passed to `do_stream()`.

Returns: A GenerateResult object.

Method `do_generate()`**:** Generate text (non-streaming). Abstract method.

Usage:

```
LanguageModelV1$do_generate(params)
```

Arguments:

`params` A list of call options.

Returns: A GenerateResult object.

Method `do_stream()`**:** Generate text (streaming). Abstract method.

Usage:

```
LanguageModelV1$do_stream(params, callback)
```

Arguments:

`params` A list of call options.

`callback` Callback invoked with `(text, done)`.

Returns: A GenerateResult object.

Method `format_tool_result()`**:** Format a tool execution result for the provider API.

Usage:

```
LanguageModelV1$format_tool_result(tool_call_id, tool_name, result_content)
```

Arguments:

`tool_call_id` The tool call identifier.

`tool_name` The tool name.

`result_content` The tool execution result.

Returns: A provider-specific tool result message.

Method `get_history_format()`**:** Get the message history format used by this model.

Usage:

```
LanguageModelV1$get_history_format()
```

Returns: A character string such as `"openai"` or `"anthropic"`.

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

Usage:

```
LanguageModelV1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`list_capability_models` *List Capability Models*

Description

List configured capability model routes.

Usage

```
list_capability_models()
```

Value

A data frame with capability route names and model settings.

list_context_handles *List Available Context Handles*

Description

Builds compact handles for live session context without embedding full object payloads in the prompt.

Usage

```
list_context_handles(session, state = NULL, persist = TRUE)
```

Arguments

session	A ChatSession or SharedSession.
state	Optional normalized context state. Defaults to the session state.
persist	Logical; whether to persist the handle list into the session context state.

Value

A list of compact context handle records.

list_models *List Models for Provider*

Description

Returns a data frame of models for the specified provider based on the static configuration. Includes enriched metadata when available (context window, pricing, capabilities).

Usage

```
list_models(provider = NULL)
```

Arguments

provider	The name of the provider (e.g., "stepfun"). If NULL, all providers are listed.

Value

A data frame containing model details.

list_r_objects *List Live R Objects*

Description

List live objects visible in the session environment.

Usage

```
list_r_objects(
  session = NULL,
  envir = NULL,
  pattern = NULL,
  include_hidden = FALSE,
  limit = 50L,
  scope = c("session", "workspace", "all")
)
```

Arguments

session	Optional ChatSession or SharedSession.
envir	Optional environment. Ignored when session is provided.
pattern	Optional regex pattern used to filter names.
include_hidden	Logical; whether to include names starting with ".".
limit	Maximum number of rows to return.
scope	One of "session", "workspace", or "all". Defaults to "session" for compatibility. "workspace" lists .GlobalEnv; "all" lists session environment objects followed by .GlobalEnv objects and includes a location column.

Value

A data frame with object metadata.

load_chat_session *Load a Chat Session*

Description

Load a previously saved ChatSession from a file.

Usage

```
load_chat_session(path, tools = NULL, hooks = NULL, registry = NULL)
```

Arguments

path	File path to load from (.rds or .json).
tools	Optional list of Tool objects (tools are not saved, must be re-provided).
hooks	Optional HookHandler object.
registry	Optional ProviderRegistry.

Value

A ChatSession object with restored state.

Examples

```
if (interactive()) {
  # Load a saved session
  chat <- load_chat_session("my_session.rds", tools = my_tools)

  # Continue where you left off
  response <- chat$send("Let's continue our discussion")
}
```

Middleware *Middleware (Base Class)*

Description

Defines a middleware that can intercept and modify model operations.

Public fields

name A descriptive name for this middleware.

Methods

Public methods:

- Middleware$transform_params()
- Middleware$wrap_generate()
- Middleware$wrap_stream()
- Middleware$clone()

Method transform_params(): Transform parameters before calling the model.

Usage:

```
Middleware$transform_params(params, type, model)
```

Arguments:

params The original call parameters.

`type` Either "generate" or "stream".
`model` The model being called.

Returns: The transformed parameters.

Method `wrap_generate()`**:** Wrap the generate operation.

Usage:

```
Middleware$wrap_generate(do_generate, params, model)
```

Arguments:

`do_generate` A function that calls the model's do_generate.
`params` The (potentially transformed) parameters.
`model` The model being called.

Returns: The result of the generation.

Method `wrap_stream()`**:** Wrap the stream operation.

Usage:

```
Middleware$wrap_stream(do_stream, params, model, callback)
```

Arguments:

`do_stream` A function that calls the model's do_stream.
`params` The (potentially transformed) parameters.
`model` The model being called.
`callback` The streaming callback function.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`migrate_pattern` *Migrate Legacy Code*

Description

Provides migration guidance for legacy code patterns.

Usage

```
migrate_pattern(pattern)
```

Arguments

`pattern`	The legacy pattern to migrate from.

Value

A list with old_pattern, new_pattern, and example.

Examples

```
if (interactive()) {
# Get migration guidance for ChatSession
guidance <- migrate_pattern("ChatSession")
cat(guidance$example)
}
```

`model` *Model Shortcut*

Description

Shortcut for default model configuration. Call with no arguments to read the current default model, or pass a model to update it. This is equivalent to calling `get_model()` and `set_model()` directly. Runtime options can be supplied without `new` to update the current default model's options.

Usage

```
model(new, ...)
```

Arguments

`new`	Optional model identifier string or `LanguageModelV1` object.
`...`	Runtime options forwarded to `set_model()`.

Value

When `new` is missing, returns the current default model. Otherwise invisibly returns the previous default model.

Examples

```
model()
model("openai:gpt-4o-mini")
model(NULL)
```

model_defaults *Default Model Configuration*

Description

Utilities for reading and updating the package-wide default language model. High-level helpers that accept `model = NULL`, including `generate_text()`, `stream_text()`, `ChatSession$new()`, `create_chat_session()`, `auto_fix()`, and the knitr `{ai}` engine, use this default when no explicit model is supplied.

multimodal *Multimodal Helpers*

Description

Helper functions for constructing multimodal messages (text and images).

object_peek *Peek at a Live R Object*

Description

Inspect a live object through the existing semantic inspection adapters.

Usage

```
object_peek(
  session,
  name,
  detail = c("summary", "structure", "sample", "full")
)
```

Arguments

`session`	A `ChatSession` or `SharedSession`.
`name`	Object name in the session environment.
`detail`	One of `"summary"`, `"structure"`, `"sample"`, or `"full"`.

Value

A compact inspection string.

`ObjectStrategy` *Object Strategy*

Description

Object Strategy

Object Strategy

Details

Strategy for generating structured objects based on a JSON Schema. This strategy instructs the LLM to produce valid JSON matching the schema, and handles parsing and validation of the output.

Super class

`aisdk::OutputStrategy` -> `ObjectStrategy`

Public fields

`schema` The schema definition (from z_object, etc.).

`schema_name` Human-readable name for the schema.

Methods

Public methods:

- `ObjectStrategy$new()`
- `ObjectStrategy$get_instruction()`
- `ObjectStrategy$validate()`
- `ObjectStrategy$clone()`

Method `new()`**:** Initialize the ObjectStrategy.

Usage:

```
ObjectStrategy$new(schema, schema_name = "json_schema")
```

Arguments:

`schema` A schema object created by z_object, z_array, etc.

`schema_name` An optional name for the schema (default: "json_schema").

Method `get_instruction()`**:** Generate the instruction for the LLM to output valid JSON.

Usage:

```
ObjectStrategy$get_instruction()
```

Returns: A character string with the prompt instruction.

Method `validate()`**:** Validate and parse the LLM output as JSON.

Usage:

```
ObjectStrategy$validate(text, is_final = FALSE)
```

Arguments:

`text` The raw text output from the LLM.

`is_final` Logical, TRUE if this is the final output.

Returns: The parsed R object (list), or NULL if parsing fails.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`openai_code_interpreter_tool` *OpenAI built-in code_interpreter tool*

Description

Give the model an OpenAI-hosted Python sandbox.

Usage

```
openai_code_interpreter_tool(container = list(type = "auto"))
```

Arguments

`container`	Container spec. Defaults to `list(type = "auto")` which lets OpenAI provision an ephemeral container. Pass a string container ID (e.g. `"cntr_abc123"`) to attach to a specific container, or a list for advanced config (e.g. `list(type = "auto", memory_limit = "4g")`).

Value

A plain list ready to drop into `tools = list(...)`.

Examples

```
## Not run:
generate_text(
  model,
  "Solve 3x + 11 = 14 step by step.",
  tools = list(openai_code_interpreter_tool())
)

## End(Not run)
```

openai_computer_use_tool

OpenAI built-in computer_use tool

Description

Browser / desktop control. The model issues actions (click, type, scroll, screenshot) which the caller's harness executes against a virtual computer.

Usage

```
openai_computer_use_tool(
  display_width = NULL,
  display_height = NULL,
  environment = c("browser", "mac", "windows", "ubuntu")
)
```

Arguments

display_width, display_height	Integer screen dimensions in pixels. Required by the API on most environments.
environment	One of "browser", "mac", "windows", "ubuntu".

Value

A plain list ready to drop into tools = list(...).

openai_file_search_tool

OpenAI built-in file_search tool

Description

Configure the Responses API's hosted RAG over one or more vector stores you've uploaded to OpenAI.

Usage

```
openai_file_search_tool(
  vector_store_ids,
  max_num_results = NULL,
  ranking_options = NULL
)
```

Arguments

`vector_store_ids`
: Character vector of vector-store IDs to search. Required, non-empty.

`max_num_results`
: Optional integer cap on how many chunks the model sees per call.

`ranking_options`
: Optional named list for advanced ranking config (e.g. `list(ranker = "auto", score_threshold = 0.5)`).

Value

A plain list ready to drop into `tools = list(...)`.

Examples

```
## Not run:
generate_text(
  model,
  "What does the design doc say about caching?",
  tools = list(openai_file_search_tool(c("vs_abc123")))
)

## End(Not run)
```

---

openai_hosted_mcp_tool *OpenAI hosted MCP tool*

Description

Give the model access to a remote MCP server, executed server-side by OpenAI. This is independent of aisdk's local MCP client (`R/mcp_client.R`) — that one connects from R, this one tells OpenAI to connect on the model's behalf.

Usage

```
openai_hosted_mcp_tool(
  server_label,
  server_url = NULL,
  connector_id = NULL,
  server_description = NULL,
  allowed_tools = NULL,
  require_approval = NULL,
  authorization = NULL,
  headers = NULL
)
```

Arguments

server_label	Required label used to identify the server in tool calls.
server_url	URL of a custom MCP server. Exactly one of server_url or connector_id is required.
connector_id	OpenAI-managed connector ID (e.g. "connector_gmail").
server_description	Optional human-readable description.
allowed_tools	Optional character vector of tool names to expose, or a list filter (e.g. list(read_only = TRUE)).
require_approval	"always", "never", or a list filter (e.g. list(always = list(tool_names = c("send_email")))).
authorization	Optional OAuth access token.
headers	Optional named list of HTTP headers sent to the MCP server.

Value

A plain list ready to drop into tools = list(...).

Examples

```
## Not run:
generate_text(
  model,
  "Roll 2d4+1",
  tools = list(openai_hosted_mcp_tool(
    server_label = "dmcp",
    server_url = "https://dmcp-server.deno.dev/sse",
    allowed_tools = "roll",
    require_approval = "never"
  ))
)

## End(Not run)
```

---

openai_web_search_tool

OpenAI built-in web_search tool

Description

Build the configuration for OpenAI's model-hosted web search, used via the Responses API. The model decides whether to search based on the prompt.

Usage

```
openai_web_search_tool(
  allowed_domains = NULL,
  user_location = NULL,
  search_context_size = NULL,
  type = c("web_search", "web_search_preview")
)
```

Arguments

allowed_domains	Optional character vector of domains the search is restricted to. Forwarded as `filters.allowed_domains`.
user_location	Optional named list with any of `city`, `country`, `region`, `timezone` for localized results.
search_context_size	Optional `"low"`, `"medium"` (API default), or `"high"` — controls how much retrieved context the model is given.
type	The tool type identifier. Defaults to `"web_search"`; pass `"web_search_preview"` for the older preview integration.

Value

A plain list ready to drop into `tools = list(...)`.

Examples

```
## Not run:
model <- create_openai()$responses_model("gpt-5")
generate_text(
  model,
  "Latest material AAPL news?",
  tools = list(openai_web_search_tool(search_context_size = "high"))
)

## End(Not run)
```

OpenAIProvider *OpenAI Provider Class*

Description

Provider class for OpenAI. Can create language and embedding models.

Public fields

`specification_version` Provider spec version.

Methods

Public methods:

- OpenAIProvider$new()
- OpenAIProvider$language_model()
- OpenAIProvider$responses_model()
- OpenAIProvider$model()
- OpenAIProvider$smart_model()
- OpenAIProvider$embedding_model()
- OpenAIProvider$image_model()
- OpenAIProvider$create_conversation()
- OpenAIProvider$get_conversation()
- OpenAIProvider$delete_conversation()
- OpenAIProvider$clone()

Method `new()`**:** Initialize the OpenAI provider.

Usage:

```
OpenAIProvider$new(
  api_key = NULL,
  base_url = NULL,
  organization = NULL,
  project = NULL,
  headers = NULL,
  name = NULL,
  timeout_seconds = NULL,
  total_timeout_seconds = NULL,
  first_byte_timeout_seconds = NULL,
  connect_timeout_seconds = NULL,
  idle_timeout_seconds = NULL,
  disable_stream_options = FALSE,
  api_format = c("auto", "chat", "responses")
)
```

Arguments:

`api_key` OpenAI API key. Defaults to OPENAI_API_KEY env var.

`base_url` Base URL for API calls. Defaults to https://api.openai.com/v1.

`organization` Optional OpenAI organization ID.

`project` Optional OpenAI project ID.

`headers` Optional additional headers.

`name` Optional provider name override (for compatible APIs).

`timeout_seconds` Legacy alias for `total_timeout_seconds`.

`total_timeout_seconds` Optional total request timeout in seconds for API calls.

`first_byte_timeout_seconds` Optional time-to-first-byte timeout in seconds for API calls.

`connect_timeout_seconds` Optional connection-establishment timeout in seconds for API calls.

`idle_timeout_seconds` Optional stall timeout in seconds for API calls.

`disable_stream_options` Disable stream_options parameter (for providers that don't support it).

`api_format` Default API surface for `smart_model()` / `model()`: "auto" (route reasoning models to Responses, others to Chat — the canonical OpenAI behavior), "chat" (always Chat Completions — useful for proxies that don't expose /responses, or that surface reasoning models like gpt-5.x via /chat/completions), or "responses" (always Responses API). The explicit `language_model()` and `responses_model()` methods continue to ignore this setting.

Method `language_model()`**:** Create a language model (always Chat Completions API).

Usage:

```
OpenAIProvider$language_model(model_id = Sys.getenv("OPENAI_MODEL", "gpt-4o"))
```

Arguments:

`model_id` The model ID (e.g., "gpt-4o", "gpt-4o-mini").

Returns: An OpenAILanguageModel object.

Method `responses_model()`**:** Create a language model using the Responses API.

Usage:

```
OpenAIProvider$responses_model(model_id)
```

Arguments:

`model_id` The model ID (e.g., "o1", "o3-mini", "gpt-4o").

Details: The Responses API is designed for:

- Models with built-in reasoning (o1, o3 series)
- Stateful multi-turn conversations (server maintains history)
- Advanced features like structured outputs

The model maintains conversation state internally via response IDs. Call `model$reset()` to start a fresh conversation.

Returns: An OpenAIResponsesLanguageModel object.

Method `model()`**:** Default-route factory. Picks chat vs responses based on the `api_format` set in `create_openai()`. Recommended entry point for callers that don't care which surface is used.

Usage:

```
OpenAIProvider$model(model_id = Sys.getenv("OPENAI_MODEL", "gpt-4o"))
```

Arguments:

`model_id` The model ID. Defaults to `OPENAI_MODEL` env var.

Returns: A LanguageModelV1 instance.

Method `smart_model()`**:** Smart model factory that selects the API surface.

Usage:

```
OpenAIProvider$smart_model(
  model_id,
  api_format = private$config$api_format %||% "auto"
)
```

Arguments:

`model_id` The model ID.

`api_format` API format: "auto" (default — reasoning → Responses, others → Chat), "chat", or "responses". Defaults to the `api_format` passed to `create_openai()`.

Details: When `api_format = "auto"`, the method picks:

- Responses API for reasoning models (o1, o3, gpt-5, ...)
- Chat Completions API for everything else

Override per-call when the provider's default doesn't match the specific model you're about to use.

Returns: A language model object (either OpenAILanguageModel or OpenAIResponsesLanguageModel).

Method `embedding_model()`**:** Create an embedding model.

Usage:

```
OpenAIProvider$embedding_model(model_id = "text-embedding-3-small")
```

Arguments:

`model_id` The model ID (e.g., "text-embedding-3-small").

Returns: An OpenAIEmbeddingModel object.

Method `image_model()`**:** Create an image model.

Usage:

```
OpenAIProvider$image_model(
  model_id = Sys.getenv("OPENAI_IMAGE_MODEL", "gpt-image-2")
)
```

Arguments:

`model_id` The model ID (e.g., "gpt-image-2", "gpt-image-1.5").

Returns: An OpenAIImageModel object.

Method `create_conversation()`**:** Create a server-side conversation object via `POST /v1/conversations`. Returns the parsed response, including the conversation `id` you can pass as `conversation = "conv_..."` to `generate_text()` / `stream_text()` so OpenAI manages the message history server-side instead of you sending the full transcript each turn.

Usage:

```
OpenAIProvider$create_conversation(items = NULL, metadata = NULL)
```

Arguments:

`items` Optional list of initial conversation items, each shaped like `list(type = "message", role = "user", content = "Hello!")`.

`metadata` Optional named list (up to 16 keys, values stringified).

Returns: Parsed response list with at least `id`, `object`, `created_at`, `metadata`.

Method `get_conversation()`**:** Retrieve a conversation object by id.

Usage:

```
OpenAIProvider$get_conversation(conversation_id)
```

Arguments:

`conversation_id` Conversation id returned from `create_conversation()`.

Returns: Parsed response list.

Method `delete_conversation()`**:** Delete a conversation object by id. Server-side history is irrecoverable after this call.

Usage:

```
OpenAIProvider$delete_conversation(conversation_id)
```

Arguments:

`conversation_id` Conversation id returned from `create_conversation()`.

Returns: Parsed response list (typically `list(id, object, deleted)`).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`OutputStrategy` *Output Strategy Interface*

Description

Output Strategy Interface

Output Strategy Interface

Details

Abstract R6 class defining the interface for output strategies. Subclasses must implement `get_instruction()` and `validate()`.

Methods

Public methods:

- `OutputStrategy$new()`
- `OutputStrategy$get_instruction()`
- `OutputStrategy$validate()`
- `OutputStrategy$clone()`

Method `new()`**:** Initialize the strategy.

Usage:

```
OutputStrategy$new()
```

Method `get_instruction()`**:** Get the system prompt instruction for this strategy.

Usage:

```
OutputStrategy$get_instruction()
```

Returns: A character string with instructions for the LLM.

Method `validate()`**:** Parse and validate the output text.

Usage:

```
OutputStrategy$validate(text, is_final = FALSE)
```

Arguments:

`text` The raw text output from the LLM.

`is_final` Logical, TRUE if this is the final output (not streaming).

Returns: The parsed and validated object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`print.GenerateObjectResult` *Print GenerateObjectResult*

Description

Print GenerateObjectResult

Usage

```
## S3 method for class 'GenerateObjectResult'
print(x, ...)
```

Arguments

`x`	A GenerateObjectResult object.
`...`	Additional arguments (ignored).

print.z_schema *Print Method for z_schema*

Description

Pretty print a z_schema object.

Usage

```
## S3 method for class 'z_schema'
print(x, ...)
```

Arguments

x	A z_schema object.
...	Additional arguments (ignored).

print_migration_guide *Print Migration Guide*

Description

Print a comprehensive migration guide for upgrading to the new SDK version.

Usage

```
print_migration_guide(verbose = TRUE)
```

Arguments

verbose	Include detailed examples. Default TRUE.

Value

Invisible NULL (prints to console).

`project_memory` *Project Memory System*

Description

Long-term memory storage for AI agents using SQLite. Stores successful code snippets, error fixes, and execution history for RAG (Retrieval Augmented Generation) and learning from past interactions.

Factory function to create or connect to a project memory database.

Usage

```
project_memory(project_root = tempdir(), db_name = "memory.sqlite")
```

Arguments

`project_root`	Project root directory.
`db_name`	Database filename.

Value

A ProjectMemory object.

Examples

```
if (interactive()) {
# Create memory for current project
memory <- project_memory()

# Store a successful code snippet
memory$store_snippet(
  code = "df %>% filter(x > 0) %>% summarize(mean = mean(y))",
  description = "Filter and summarize data",
  tags = c("dplyr", "summarize")
)

# Store an error fix
memory$store_fix(
  original_code = "mean(df$x)",
  error = "argument is not numeric or logical",
  fixed_code = "mean(as.numeric(df$x), na.rm = TRUE)",
  fix_description = "Convert to numeric and handle NAs"
)

# Search for relevant snippets
memory$search_snippets("summarize")
}
```

`ProjectMemory` *Project Memory Class*

Description

R6 class for managing persistent project memory using SQLite. Stores code snippets, error fixes, and execution graphs for resuming failed long-running jobs.

Public fields

`db_path` Path to the SQLite database file.

`project_root` Root directory of the project.

Methods

Public methods:

- `ProjectMemory$new()`
- `ProjectMemory$store_snippet()`
- `ProjectMemory$store_fix()`
- `ProjectMemory$find_similar_fix()`
- `ProjectMemory$search_snippets()`
- `ProjectMemory$store_workflow_node()`
- `ProjectMemory$update_node_status()`
- `ProjectMemory$get_workflow()`
- `ProjectMemory$get_resumable_nodes()`
- `ProjectMemory$store_conversation()`
- `ProjectMemory$get_conversation()`
- `ProjectMemory$store_review()`
- `ProjectMemory$store_review_artifact()`
- `ProjectMemory$get_review()`
- `ProjectMemory$get_review_artifact()`
- `ProjectMemory$get_review_runtime_record()`
- `ProjectMemory$get_reviews_for_file()`
- `ProjectMemory$record_review_saveback()`
- `ProjectMemory$update_execution_result()`
- `ProjectMemory$append_review_event()`
- `ProjectMemory$update_review_status()`
- `ProjectMemory$get_pending_reviews()`
- `ProjectMemory$stats()`
- `ProjectMemory$clear()`
- `ProjectMemory$print()`
- `ProjectMemory$clone()`

Method `new()`**:** Create or connect to a project memory database.

Usage:

```
ProjectMemory$new(project_root = tempdir(), db_name = "memory.sqlite")
```

Arguments:

`project_root` Project root directory. Defaults to `tempdir()`.

`db_name` Database filename. Defaults to "memory.sqlite".

Returns: A new ProjectMemory object.

Method `store_snippet()`**:** Store a successful code snippet for future reference.

Usage:

```
ProjectMemory$store_snippet(
  code,
  description = NULL,
  tags = NULL,
  context = NULL
)
```

Arguments:

`code` The R code that was executed successfully.

`description` Optional description of what the code does.

`tags` Optional character vector of tags for categorization.

`context` Optional context about when/why this code was used.

Returns: The ID of the stored snippet.

Method `store_fix()`**:** Store an error fix for learning.

Usage:

```
ProjectMemory$store_fix(
  original_code,
  error,
  fixed_code,
  fix_description = NULL
)
```

Arguments:

`original_code` The code that produced the error.

`error` The error message.

`fixed_code` The corrected code.

`fix_description` Description of what was fixed.

Returns: The ID of the stored fix.

Method `find_similar_fix()`**:** Find a similar fix from memory.

Usage:

```
ProjectMemory$find_similar_fix(error)
```

Arguments:

`error` The error message to match.

Returns: A list with the fix details, or NULL if not found.

Method `search_snippets()`**:** Search for relevant code snippets.

Usage:

```
ProjectMemory$search_snippets(query, limit = 10)
```

Arguments:

`query` Search query (matches description, tags, or code).

`limit` Maximum number of results.

Returns: A data frame of matching snippets.

Method `store_workflow_node()`**:** Store execution graph node for workflow persistence.

Usage:

```
ProjectMemory$store_workflow_node(
  workflow_id,
  node_id,
  node_type,
  code,
  status = "pending",
  result = NULL,
  dependencies = NULL
)
```

Arguments:

`workflow_id` Unique identifier for the workflow.

`node_id` Unique identifier for this node.

`node_type` Type of node (e.g., "transform", "model", "output").

`code` The code for this node.

`status` Node status ("pending", "running", "completed", "failed").

`result` Optional serialized result.

`dependencies` Character vector of node IDs this depends on.

Returns: The database row ID.

Method `update_node_status()`**:** Update workflow node status.

Usage:

```
ProjectMemory$update_node_status(workflow_id, node_id, status, result = NULL)
```

Arguments:

`workflow_id` Workflow identifier.

`node_id` Node identifier.

`status` New status.

`result` Optional result to store.

Method `get_workflow()`**:** Get workflow state for resuming.

Usage:

```
ProjectMemory$get_workflow(workflow_id)
```

Arguments:

`workflow_id` Workflow identifier.

Returns: A list with workflow nodes and their states.

Method `get_resumable_nodes()`**:** Resume a failed workflow from the last successful point.

Usage:

```
ProjectMemory$get_resumable_nodes(workflow_id)
```

Arguments:

`workflow_id` Workflow identifier.

Returns: List of node IDs that need to be re-executed.

Method `store_conversation()`**:** Store a conversation turn for context.

Usage:

```
ProjectMemory$store_conversation(session_id, role, content, metadata = NULL)
```

Arguments:

`session_id` Session identifier.

`role` Message role ("user", "assistant", "system").

`content` Message content.

`metadata` Optional metadata list.

Method `get_conversation()`**:** Get conversation history for a session.

Usage:

```
ProjectMemory$get_conversation(session_id, limit = 100)
```

Arguments:

`session_id` Session identifier.

`limit` Maximum number of messages.

Returns: A data frame of conversation messages.

Method `store_review()`**:** Store or update a human review for an AI-generated chunk.

Usage:

```
ProjectMemory$store_review(
  chunk_id,
  file_path,
  chunk_label,
  prompt,
  response,
  status = "pending",
  ai_agent = NULL,
  uncertainty = NULL,
  session_id = NULL,
  review_mode = NULL,
  runtime_mode = NULL,
  artifact_json = NULL,
  execution_status = NULL,
```

```
  execution_output = NULL,
  final_code = NULL,
  error_message = NULL
)
```

Arguments:

`chunk_id` Unique identifier for the chunk.

`file_path` Path to the source file.

`chunk_label` Chunk label from knitr.

`prompt` The prompt sent to the AI.

`response` The AI's response.

`status` Review status ("pending", "approved", "rejected").

`ai_agent` Optional agent name.

`uncertainty` Optional uncertainty level.

`session_id` Optional session identifier for transcript/provenance.

`review_mode` Optional normalized review mode.

`runtime_mode` Optional normalized runtime mode.

`artifact_json` Optional JSON review artifact payload.

`execution_status` Optional execution state.

`execution_output` Optional execution output text.

`final_code` Optional finalized executable code.

`error_message` Optional execution or generation error.

Returns: The database row ID.

Method `store_review_artifact()`**:** Store structured review artifact metadata for a chunk.

Usage:

```
ProjectMemory$store_review_artifact(
  chunk_id,
  artifact,
  session_id = NULL,
  review_mode = NULL,
  runtime_mode = NULL
)
```

Arguments:

`chunk_id` Chunk identifier.

`artifact` A serializable list representing the review artifact.

`session_id` Optional session identifier.

`review_mode` Optional normalized review mode.

`runtime_mode` Optional normalized runtime mode.

Returns: Invisible TRUE.

Method `get_review()`**:** Get a review by chunk ID.

Usage:

```
ProjectMemory$get_review(chunk_id)
```

Arguments:

`chunk_id` Chunk identifier.

Returns: A list with review details, or NULL if not found.

Method `get_review_artifact()`**:** Get a parsed review artifact by chunk ID.

Usage:

```
ProjectMemory$get_review_artifact(chunk_id)
```

Arguments:

`chunk_id` Chunk identifier.

Returns: A list artifact, or NULL if none is stored.

Method `get_review_runtime_record()`**:** Get a review together with its parsed artifact.

Usage:

```
ProjectMemory$get_review_runtime_record(chunk_id)
```

Arguments:

`chunk_id` Chunk identifier.

Returns: A list with `review` and `artifact`, or NULL if not found.

Method `get_reviews_for_file()`**:** Get all reviews for a given source file.

Usage:

```
ProjectMemory$get_reviews_for_file(file_path)
```

Arguments:

`file_path` Source document path.

Returns: A data frame of reviews ordered by updated time.

Method `record_review_saveback()`**:** Record a saveback lifecycle event for one or more chunk reviews.

Usage:

```
ProjectMemory$record_review_saveback(
  chunk_ids,
  source_path,
  html_path = NULL,
  status = "requested",
  rerendered = FALSE,
  message = NULL
)
```

Arguments:

`chunk_ids` Character vector of chunk identifiers.

`source_path` Source document path.

`html_path` Optional rendered HTML path.

`status` Saveback status string.

`rerendered` Whether a rerender occurred.

`message` Optional message.

Returns: Invisible TRUE.

Method `update_execution_result()`**:** Update execution result fields for a chunk review.

Usage:

```
ProjectMemory$update_execution_result(
  chunk_id,
  execution_status,
  execution_output = NULL,
  final_code = NULL,
  error_message = NULL
)
```

Arguments:

`chunk_id` Chunk identifier.

`execution_status` Execution state string.

`execution_output` Optional execution output text.

`final_code` Optional finalized executable code.

`error_message` Optional execution error.

Returns: Invisible TRUE.

Method `append_review_event()`**:** Append an audit event for a reviewed chunk.

Usage:

```
ProjectMemory$append_review_event(chunk_id, event_type, payload = NULL)
```

Arguments:

`chunk_id` Chunk identifier.

`event_type` Event type string.

`payload` Optional serializable payload list.

Returns: The database row ID.

Method `update_review_status()`**:** Update review status.

Usage:

```
ProjectMemory$update_review_status(chunk_id, status)
```

Arguments:

`chunk_id` Chunk identifier.

`status` New status ("approved" or "rejected").

Method `get_pending_reviews()`**:** Get pending reviews, optionally filtered by file.

Usage:

```
ProjectMemory$get_pending_reviews(file_path = NULL)
```

Arguments:

`file_path` Optional file path filter.

Returns: A data frame of pending reviews.

Method `stats()`**:** Get memory statistics.

Usage:

```
ProjectMemory$stats()
```

Returns: A list with counts and sizes.

Method `clear()`**:** Clear all memory (use with caution).

Usage:

```
ProjectMemory$clear(confirm = FALSE)
```

Arguments:

`confirm` Must be TRUE to proceed.

Method `print()`**:** Print method for ProjectMemory.

Usage:

```
ProjectMemory$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`provider_custom` *Custom Provider Factory*

Description

A dynamic factory for creating custom provider instances. This allows users to instantiate a model provider at runtime by configuring the endpoint (`base_url`), credentials (`api_key`), network protocol/routing (`api_format`), and specific capabilities (`use_max_completion_tokens`), without writing a new Provider class.

`ProviderRegistry` *Provider Registry*

Description

Manages registered providers and allows accessing models by ID.

Methods

Public methods:

- `ProviderRegistry$new()`
- `ProviderRegistry$register()`
- `ProviderRegistry$language_model()`
- `ProviderRegistry$embedding_model()`
- `ProviderRegistry$image_model()`
- `ProviderRegistry$list_providers()`
- `ProviderRegistry$clone()`

Method `new()`**:** Initialize the registry.

Usage:

```
ProviderRegistry$new(separator = ":")
```

Arguments:

`separator` The separator between provider and model IDs (default: ":").

Method `register()`**:** Register a provider.

Usage:

```
ProviderRegistry$register(id, provider)
```

Arguments:

`id` The provider ID (e.g., "openai").

`provider` The provider object (must have `language_model` method).

Method `language_model()`**:** Get a language model by ID.

Usage:

```
ProviderRegistry$language_model(id)
```

Arguments:

`id` Model ID in the format "provider:model" (e.g., "openai:gpt-4o").

Returns: A LanguageModelV1 object.

Method `embedding_model()`**:** Get an embedding model by ID.

Usage:

```
ProviderRegistry$embedding_model(id)
```

Arguments:

`id` Model ID in the format "provider:model".

Returns: An EmbeddingModelV1 object.

Method `image_model()`**:** Get an image model by ID.

Usage:

```
ProviderRegistry$image_model(id)
```

Arguments:

`id` Model ID in the format "provider:model".

Returns: An ImageModelV1 object.

Method `list_providers()`**:** List all registered provider IDs.

Usage:

```
ProviderRegistry$list_providers()
```

Returns: A character vector of provider IDs.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`r_context_tools` *R Context Tools*

Description

Read-only helpers and built-in tools for inspecting live R session objects, functions, documentation, and source code.

`r_introspect_tools` *R Introspection Tools for Autonomous Debugging*

Description

General-purpose primitives that let a console Agent enrich diagnostic context on its own when R's `geterrmessage()` / `last.warning` / `traceback()` snapshots are incomplete.

These tools are intentionally **broad** (not problem-specific). Domain tactics – e.g. where to look for an install log, how to interpret an Rcpp compilation error – live in the `r-debug` skill (`inst/skills/r-debug/`). The tools provide eyes and hands; the skill provides the playbook.

Provided tools:

- `r_eval`: run R code in an isolated subprocess, capture stdout, stderr (including from grandchild processes like compilers and `install.packages` subprocesses), messages, warnings, value, error.
- `r_session_state`: structured snapshot of the live R session (`.libPaths()`, repos, search path, key env vars, options, sessionInfo).

register_provider *Register a Provider Factory*

Description

Register an additional provider factory so it can be resolved through the default registry's `provider:model` syntax. Intended for companion packages (such as **aisdk.providers**) that ship OpenAI-compatible providers and register them from their `.onLoad` hook, e.g. `aisdk::register_provider("deepseek", function() create_deepseek())`.

Registration is load-order independent: the factory is replayed into the default registry whether it is registered before or after the registry is first built.

Usage

```
register_provider(id, factory)
```

Arguments

`id`	The provider ID (e.g. "deepseek").
`factory`	A zero-argument function returning a provider object, or a function of one argument (`model_id`) returning a language model.

Value

Invisibly `TRUE`.

reload_env *Reload project-level environment variables*

Description

Forces R to re-read the .Renviron file without restarting the session. This is useful when you've modified .Renviron and don't want to restart R.

Usage

```
reload_env(path = ".Renviron")
```

Arguments

`path`	Path to .Renviron file (default: project root)

Value

Invisible TRUE if successful

Examples

```
if (interactive()) {
# Reload environment after modifying .Renviron
reload_env()
# Now use the new keys
Sys.getenv("OPENAI_API_KEY")
}
```

`render_text` *Render Markdown Text*

Description

Render markdown-formatted text in the console with beautiful styling. This function uses the same rendering engine as the streaming output, supporting headers, lists, code blocks, and other markdown elements.

Usage

```
render_text(text)
```

Arguments

`text`	A character string containing markdown text, or a GenerateResult object.

Value

NULL (invisibly)

Examples

```
if (interactive()) {
# Render simple text
render_text("# Hello\n\nThis is **bold** text.")

# Render with code block
render_text("Here is some R code:\n\n```r\nx <- 1:10\nmean(x)\n```")
}
```

request_authorization *Request User Authorization (HITL)*

Description

Pauses execution and prompts the user for permission to execute a potentially risky action. Supports console environments via `readline`.

Usage

```
request_authorization(action, risk_level = "YELLOW")
```

Arguments

`action`	Character string describing the action the Agent wants to perform.
`risk_level`	Character string. One of "GREEN", "YELLOW", "RED".

Value

Logical TRUE if user permits, otherwise throws an error with the rejection reason.

---

resolve_model_for_capability
Resolve Model For Capability

Description

Resolve and validate the model selected for a capability route.

Usage

```
resolve_model_for_capability(
  capability,
  explicit_model = NULL,
  type = c("language", "embedding", "image"),
  required_model_capabilities = NULL,
  session = NULL,
  registry = NULL,
  fallback_model = NULL,
  default_model = NULL
)
```

Arguments

capability	Capability route name.
explicit_model	Optional explicit model override.
type	Expected model type: "language", "embedding", or "image".
required_model_capabilities	Optional required capability flags.
session	Optional ChatSession; session routes override global routes.
registry	Optional provider registry.
fallback_model	Optional model used when no route is configured.
default_model	Optional final fallback. If omitted for language models, the package default model is used.

Value

A resolved model object.

---

resolve_r_binding *Resolve an R Binding*

Description

Resolve a symbol-like name against the live session environment, attached search path, and loaded namespaces.

Usage

```
resolve_r_binding(
  name,
  package = NULL,
  session = NULL,
  envir = NULL,
  scope = c("session", "workspace", "all"),
  prefer = c("auto", "object", "function")
)
```

Arguments

name	Symbol-like name to resolve.
package	Optional package/namespace name to search first.
session	Optional ChatSession or SharedSession.
envir	Optional environment. Ignored when session is provided.
scope	Where to resolve live objects before the search path: "session" checks only the session environment, "workspace" checks only .GlobalEnv, and "all" checks session environment first, then .GlobalEnv.
prefer	Preferred kind: "auto", "object", or "function".

Value

A binding record list or NULL if not found.

`run_state` *Run State Helpers*

Description

Internal helpers for normalizing generation stop states.

`safe_eval` *Safe Eval with Timeout*

Description

Execute R code with a timeout to prevent infinite loops.

Usage

```
safe_eval(expr, timeout_seconds = 30, envir = parent.frame())
```

Arguments

`expr`	Expression to evaluate.
`timeout_seconds`	Maximum execution time in seconds.
`envir`	Environment for evaluation.

Value

The result or an error.

safe_parse_json *Safe JSON Parser*

Description

Parses a JSON string, attempting to repair it using `fix_json` if the initial parse fails.

Usage

```
safe_parse_json(text)
```

Arguments

`text`	A JSON string.

Value

A parsed R object (list, vector, etc.) or NULL if parsing fails even after repair.

Examples

```
safe_parse_json('{"a": 1}')
safe_parse_json('{"a": 1,')
```

safe_to_json *Safe Serialization to JSON*

Description

Standardized internal helper for JSON serialization with common defaults. Object types registered via `register_json_coercion()` (for example ggplot objects, via **aisdk.datatools**) are coerced before serialization.

Usage

```
safe_to_json(x, auto_unbox = TRUE, ...)
```

Arguments

`x`	Object to serialize.
`auto_unbox`	Whether to automatically unbox single-element vectors. Default TRUE.
`...`	Additional arguments to jsonlite::toJSON.

Value

A JSON string.

sandbox *R-Native Programmatic Sandbox*

Description

SandboxManager R6 class and utilities for building an R-native programmatic tool sandbox. Inspired by Anthropic's Programmatic Tool Calling, this module enables LLMs to write R code that batch-invokes registered tools and processes data locally (using dplyr/purrr), returning only concise results to the context.

Details

The core idea: instead of the LLM making N separate tool calls (each requiring a round-trip), it writes a single R script that loops over inputs, calls tools as ordinary R functions, filters/aggregates the results with dplyr, and `print()`s only the key findings. This dramatically reduces token usage, latency, and context window pressure.

Architecture:

```
User tools -> SandboxManager -> isolated R environment
  - tool_a()
  - tool_b()
  - dplyr::*
  - purrr::*

create_r_code_tool() -> single "execute_r_code" tool
  (registered with the LLM)
```

SandboxManager *SandboxManager Class*

Description

R6 class that manages an isolated R environment for executing LLM-generated R code. Tools are bound as callable functions within this environment, enabling the LLM to batch-invoke and process data locally.

Methods

Public methods:

- `SandboxManager$new()`
- `SandboxManager$bind_tools()`
- `SandboxManager$execute()`
- `SandboxManager$get_tool_signatures()`
- `SandboxManager$get_env()`

- SandboxManager$list_tools()
- SandboxManager$reset()
- SandboxManager$print()
- SandboxManager$clone()

Method `new()`**:** Initialize a new SandboxManager.

Usage:

```
SandboxManager$new(
  tools = list(),
  preload_packages = c("dplyr", "purrr"),
  max_output_chars = 8000,
  parent_env = NULL
)
```

Arguments:

`tools` Optional list of Tool objects to bind into the sandbox.

`preload_packages` Character vector of package names to preload into the sandbox (their exports become available). Default: c("dplyr", "purrr").

`max_output_chars` Maximum characters to capture from code output. Prevents runaway `print()` from flooding the context. Default: 8000.

`parent_env` Optional parent environment for the sandbox. When a ChatSession is available, pass `session$get_envir()` here to enable cross-step variable persistence.

Method `bind_tools()`**:** Bind Tool objects into the sandbox as callable R functions.

Usage:

```
SandboxManager$bind_tools(tools)
```

Arguments:

`tools` A list of Tool objects to bind.

Returns: Invisible self (for chaining).

Method `execute()`**:** Execute R code in the sandbox environment.

Usage:

```
SandboxManager$execute(code_str)
```

Arguments:

`code_str` A character string containing R code to execute.

Returns: A character string with captured stdout, or an error message.

Method `get_tool_signatures()`**:** Get human-readable signatures for all bound tools.

Usage:

```
SandboxManager$get_tool_signatures()
```

Returns: A character string with Markdown-formatted tool documentation.

Method `get_env()`**:** Get the sandbox environment.

Usage:

```
SandboxManager$get_env()
```

Returns: The R environment used by the sandbox.

Method `list_tools()`**:** Get list of bound tool names.

Usage:

```
SandboxManager$list_tools()
```

Returns: Character vector of tool names available in the sandbox.

Method `reset()`**:** Reset the sandbox environment (clear all user variables). Tool bindings and preloaded packages are preserved.

Usage:

```
SandboxManager$reset()
```

Method `print()`**:** Print method for SandboxManager.

Usage:

```
SandboxManager$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`scan_skills` *Scan for Skills*

Description

Convenience function to scan a directory and return a SkillRegistry. Alias for create_skill_registry().

Usage

```
scan_skills(path, recursive = FALSE)
```

Arguments

`path`	Path to scan for skills.
`recursive`	Whether to scan subdirectories. Default FALSE.

Value

A populated SkillRegistry object.

schema *Schema DSL: Lightweight JSON Schema Generator*

Description

A lightweight DSL (Domain Specific Language) for defining JSON Schema structures in R, inspired by Zod from TypeScript. Used for defining tool parameters.

schema_generator *Schema Generator*

Description

Utilities to automatically generate z_schema objects from R function signatures.

schema_to_json *Convert Schema to JSON*

Description

Convert a z_schema object to a JSON string suitable for API calls. Handles the R-specific auto_unbox issues properly.

Usage

```
schema_to_json(schema, pretty = FALSE)
```

Arguments

schema	A z_schema object created by z_* functions.
pretty	If TRUE, format JSON with indentation.

Value

A JSON string.

Examples

```
schema <- z_object(
  name = z_string(description = "User name")
)
cat(schema_to_json(schema, pretty = TRUE))
```

sdk_clear_protected_vars *Reset the Variable Registry*

Description

Clears all protected variables.

Usage

```
sdk_clear_protected_vars()
```

sdk_feature *Get Feature Flag*

Description

Get the current value of a feature flag.

Usage

```
sdk_feature(flag, default = NULL)
```

Arguments

flag	Name of the feature flag.
default	Default value if flag not set.

Value

The flag value.

Examples

```
if (interactive()) {
# Check if shared session is enabled
if (sdk_feature("use_shared_session")) {
  session <- create_shared_session(model = "openai:gpt-4o")
}
} else {
  session <- create_chat_session(model = "openai:gpt-4o")
}
```

sdk_get_var_metadata *Get Metadata for a Protected Variable*

Description

Get Metadata for a Protected Variable

Usage

```
sdk_get_var_metadata(name)
```

Arguments

name	Character string. The name of the variable.

Value

A list with metadata (locked, cost, etc.), or NULL if not protected.

sdk_is_var_locked *Check if a Variable is Locked*

Description

Check if a Variable is Locked

Usage

```
sdk_is_var_locked(name)
```

Arguments

name	Character string. The name of the variable.

Value

TRUE if the variable is protected and locked, FALSE otherwise.

sdk_list_features *List Feature Flags*

Description

List all available feature flags and their current values.

Usage

```
sdk_list_features()
```

Value

A named list of feature flags.

Examples

```
if (interactive()) {
# See all feature flags
print(sdk_list_features())
}
```

sdk_protect_var *Protect a Variable from Agent Modification*

Description

Marks a variable as protected so that the Agent cannot accidentally overwrite, shadow, or deeply copy it during sandbox execution.

Usage

```
sdk_protect_var(name, locked = TRUE, cost = "High")
```

Arguments

name	Character string. The name of the variable to protect.
locked	Logical. If TRUE, the variable cannot be assigned to by the Agent.
cost	Character string. An indicator of the variable's computation/memory cost (e.g., "High", "Medium", "Low").

Value

Invisible TRUE.

sdk_reset_features *Reset Feature Flags*

Description

Reset all feature flags to their default values.

Usage

```
sdk_reset_features()
```

Value

Invisible NULL.

sdk_set_feature *Set Feature Flag*

Description

Set a feature flag value. Use this to enable/disable SDK features.

Usage

```
sdk_set_feature(flag, value)
```

Arguments

flag	Name of the feature flag.
value	Value to set.

Value

Invisible previous value.

Examples

```
if (interactive()) {
# Disable shared session for legacy compatibility
sdk_set_feature("use_shared_session", FALSE)

# Enable legacy tool format
sdk_set_feature("legacy_tool_format", TRUE)
}
```

sdk_unprotect_var *Unprotect a Variable*

Description

Removes protection from a previously protected variable.

Usage

```
sdk_unprotect_var(name)
```

Arguments

name	Character string. The name of the variable to unprotect.

Value

Invisible TRUE.

semantic_adapter *Semantic Adapter Runtime*

Description

Internal semantic adapter protocol and registry for object-aware context construction and inspection.

session *Session Management: Stateful Chat Sessions*

Description

ChatSession R6 class for managing stateful conversations with LLMs. Provides automatic history management, persistence, and model switching.

session_event_store *Console Session Event Store*

Description

JSONL-backed session event helpers for console runs.

`set_capability_model` *Set Capability Model*

Description

Set the model used for a named capability. For example, the default chat model can remain a low-cost text model while `vision.inspect` uses a vision-capable language model and `image.generate` uses an image model.

Usage

```
set_capability_model(
  capability,
  model,
  type = "auto",
  required_model_capabilities = NULL
)
```

Arguments

`capability`	Capability route name, such as `"vision.inspect"` or a named list of routes for batch updates.
`model`	Model ID string or model object. Passing `NULL` clears the route for `capability`.
`type`	Model type for this route: `"auto"`, `"language"`, `"embedding"`, or `"image"`.
`required_model_capabilities`	Optional model capability flags required by this route, such as `"vision_input"`.

Value

Invisibly returns the previous model for the route, or the previous route list for batch updates.

Examples

```
old <- set_capability_model("vision.inspect", "openai:gpt-4o")
get_capability_model("vision.inspect")
set_capability_model("vision.inspect", old)
```

set_context_management_config *Set Context Management Configuration*

Description

Apply adaptive context-management settings to a session.

Usage

```
set_context_management_config(
  session,
  config = NULL,
  mode = NULL,
  llm_synthesis = NULL,
  synthesis_model = NULL,
  llm_synthesis_policy = NULL,
  context_window_override = NULL,
  max_output_tokens_override = NULL,
  project_memory_root = NULL,
  retrieval_providers = NULL,
  retrieval_provider_order = NULL,
  retrieval_provider_limits = NULL,
  retrieval_min_hits = NULL,
  retrieval_scoring_policy = NULL,
  retrieval_reranking = NULL,
  retrieval_reranking_model = NULL,
  retrieval_reranking_policy = NULL
)
```

Arguments

session	A ChatSession or SharedSession.
config	Optional config list created by create_context_management_config().
mode	Optional override for config mode.
llm_synthesis	Optional override for config LLM synthesis flag.
synthesis_model	Optional override for config synthesis model.
llm_synthesis_policy	Optional override for config LLM synthesis policy.
context_window_override	Optional override for context window.
max_output_tokens_override	Optional override for max output tokens.

`project_memory_root`
: Optional override for project memory root.

`retrieval_providers`
: Optional override for retrieval-provider selection.

`retrieval_provider_order`
: Optional override for provider ranking/order.

`retrieval_provider_limits`
: Optional override for per-provider limits.

`retrieval_min_hits`
: Optional override for per-provider matching thresholds.

`retrieval_scoring_policy`
: Optional override for cross-provider scoring policy.

`retrieval_reranking`
: Optional override for learned reranking enablement.

`retrieval_reranking_model`
: Optional override for the reranker model.

`retrieval_reranking_policy`
: Optional override for reranking policy.

Value

Invisible `session`.

`set_model` *Set Default Model*

Description

Sets the package-wide default language model. Pass `NULL` to restore the built-in default (`"openai:gpt-4o"` unless overridden with `options(aisdk.default_model = ...)`). If `new` is omitted and runtime options are supplied, only the runtime options are updated.

Usage

```
set_model(
  new = NULL,
  context_window = NULL,
  max_output_tokens = NULL,
  max_tokens = NULL,
  thinking = NULL,
  thinking_budget = NULL,
  reasoning_effort = NULL,
  reset_options = FALSE
)
```

Arguments

new	A model identifier string, a LanguageModelV1 object, or NULL.
context_window	Optional context-window override used by sessions created from this default model.
max_output_tokens	Optional maximum output-token metadata override.
max_tokens	Optional default generation token limit.
thinking	Optional default thinking-mode value passed to providers that support it. Logical values are normalized by each provider.
thinking_budget	Optional default thinking budget.
reasoning_effort	Optional default reasoning effort ("low", "medium", or "high").
reset_options	Logical. If TRUE, clears default runtime options.

Value

Invisibly returns the previous default model.

Examples

```
old <- set_model("deepseek:deepseek-chat")
current <- get_model()
set_model(old)
set_model(NULL)
```

---

shared_session *SharedSession: Enhanced Multi-Agent Session Management*

Description

SharedSession R6 class providing enhanced environment management, execution context tracking, and safety guardrails for multi-agent orchestration.

SharedSession *SharedSession Class*

Description

R6 class representing an enhanced session for multi-agent systems. Extends ChatSession with:

- Execution context tracking (call stack, delegation history)
- Sandboxed code execution with safety guardrails
- Variable scoping and access control
- Comprehensive tracing and observability

Super class

`aisdk::ChatSession` -> `SharedSession`

Methods

Public methods:

- `SharedSession$new()`
- `SharedSession$push_context()`
- `SharedSession$pop_context()`
- `SharedSession$get_context()`
- `SharedSession$set_global_task()`
- `SharedSession$execute_code()`
- `SharedSession$get_var()`
- `SharedSession$set_var()`
- `SharedSession$list_vars()`
- `SharedSession$summarize_vars()`
- `SharedSession$create_scope()`
- `SharedSession$delete_scope()`
- `SharedSession$trace_event()`
- `SharedSession$get_trace()`
- `SharedSession$clear_trace()`
- `SharedSession$trace_summary()`
- `SharedSession$set_access_control()`
- `SharedSession$check_permission()`
- `SharedSession$get_sandbox_mode()`
- `SharedSession$set_sandbox_mode()`
- `SharedSession$print()`
- `SharedSession$clone()`

Method `new()`**:** Initialize a new SharedSession.

Usage:

```
SharedSession$new(
  model = NULL,
  system_prompt = NULL,
  tools = NULL,
  hooks = NULL,
  max_steps = 10,
  registry = NULL,
  sandbox_mode = "strict",
  trace_enabled = TRUE
)
```

Arguments:

`model` A LanguageModelV1 object or model string ID.

`system_prompt` Optional system prompt for the conversation.

`tools` Optional list of Tool objects.

`hooks` Optional HookHandler object.

`max_steps` Maximum steps for tool execution loops. Default 10.

`registry` Optional ProviderRegistry for model resolution.

`sandbox_mode` Sandbox mode: "strict", "permissive", or "none". Default "strict".

`trace_enabled` Enable execution tracing. Default TRUE.

Method `push_context()`**:** Push an agent onto the execution stack.

Usage:

```
SharedSession$push_context(agent_name, task, parent_agent = NULL)
```

Arguments:

`agent_name` Name of the agent being activated.

`task` The task being delegated.

`parent_agent` Name of the delegating agent (or NULL for root).

Returns: Invisible self for chaining.

Method `pop_context()`**:** Pop the current agent from the execution stack.

Usage:

```
SharedSession$pop_context(result = NULL)
```

Arguments:

`result` Optional result from the completed agent.

Returns: The popped context, or NULL if stack was empty.

Method `get_context()`**:** Get the current execution context.

Usage:

```
SharedSession$get_context()
```

Returns: A list with current_agent, depth, and delegation_stack.

Method `set_global_task()`**:** Set the global task (user's original request).

Usage:

```
SharedSession$set_global_task(task)
```

Arguments:

`task` The global task description.

Returns: Invisible self for chaining.

Method `execute_code()`**:** Execute R code in a sandboxed environment.

Usage:

```
SharedSession$execute_code(
  code,
  scope = "global",
  timeout_ms = 30000,
  capture_output = TRUE
)
```

Arguments:

`code` R code to execute (character string).

`scope` Variable scope: "global", "agent", or a custom scope name.

`timeout_ms` Execution timeout in milliseconds. Default 30000.

`capture_output` Capture stdout/stderr. Default TRUE.

Returns: A list with result, output, error, and duration_ms.

Method `get_var()`**:** Get a variable from a specific scope.

Usage:

```
SharedSession$get_var(name, scope = "global", default = NULL)
```

Arguments:

`name` Variable name.

`scope` Scope name. Default "global".

`default` Default value if not found.

Returns: The variable value or default.

Method `set_var()`**:** Set a variable in a specific scope.

Usage:

```
SharedSession$set_var(name, value, scope = "global")
```

Arguments:

`name` Variable name.

`value` Variable value.

`scope` Scope name. Default "global".

Returns: Invisible self for chaining.

Method `list_vars()`**:** List variables in a scope.

Usage:

```
SharedSession$list_vars(scope = "global", pattern = NULL)
```

Arguments:

`scope` Scope name. Default "global".
`pattern` Optional pattern to filter names.

Returns: Character vector of variable names.

Method `summarize_vars()`**:** Get a summary of all variables in a scope.

Usage:

```
SharedSession$summarize_vars(scope = "global")
```

Arguments:

`scope` Scope name. Default "global".

Returns: A data frame with name, type, and size information.

Method `create_scope()`**:** Create a new variable scope.

Usage:

```
SharedSession$create_scope(scope_name, parent_scope = "global")
```

Arguments:

`scope_name` Name for the new scope.
`parent_scope` Parent scope name. Default "global".

Returns: Invisible self for chaining.

Method `delete_scope()`**:** Delete a variable scope.

Usage:

```
SharedSession$delete_scope(scope_name)
```

Arguments:

`scope_name` Name of the scope to delete.

Returns: Invisible self for chaining.

Method `trace_event()`**:** Record a trace event.

Usage:

```
SharedSession$trace_event(event_type, data = list())
```

Arguments:

`event_type` Type of event (e.g., "context_push", "code_execution").
`data` Event data as a list.

Returns: Invisible self for chaining.

Method `get_trace()`**:** Get the execution trace.

Usage:

```
SharedSession$get_trace(event_types = NULL, agent = NULL)
```

Arguments:

`event_types` Optional filter by event types.
`agent` Optional filter by agent name.

Returns: A list of trace events.

Method `clear_trace()`**:** Clear the execution trace.

Usage:

```
SharedSession$clear_trace()
```

Returns: Invisible self for chaining.

Method `trace_summary()`**:** Get trace summary statistics.

Usage:

```
SharedSession$trace_summary()
```

Returns: A list with event counts, agent activity, and timing info.

Method `set_access_control()`**:** Set access control for an agent.

Usage:

```
SharedSession$set_access_control(agent_name, permissions)
```

Arguments:

`agent_name` Agent name.
`permissions` List of permissions (read_scopes, write_scopes, tools).

Returns: Invisible self for chaining.

Method `check_permission()`**:** Check if an agent has permission for an action.

Usage:

```
SharedSession$check_permission(agent_name, action, target)
```

Arguments:

`agent_name` Agent name.
`action` Action type: "read", "write", or "tool".
`target` Target scope or tool name.

Returns: TRUE if permitted, FALSE otherwise.

Method `get_sandbox_mode()`**:** Get sandbox mode.

Usage:

```
SharedSession$get_sandbox_mode()
```

Returns: The current sandbox mode.

Method `set_sandbox_mode()`**:** Set sandbox mode.

Usage:

```
SharedSession$set_sandbox_mode(mode)
```

Arguments:

`mode` Sandbox mode: "strict", "permissive", or "none".

Returns: Invisible self for chaining.

Method `print()`**:** Print method for SharedSession.

Usage:

```
SharedSession$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Skill` *Skill Class*

Description

R6 class representing a skill with progressive loading capabilities. A Skill consists of:

- Level 1: YAML frontmatter (name, description) - always loaded
- Level 2: SKILL.md body (detailed instructions) - on demand
- Level 3: R scripts (executable code) - executed by agent

Public fields

`name` The unique name of the skill (from YAML frontmatter).

`description` A brief description of the skill (from YAML frontmatter).

`aliases` Optional aliases that should also trigger this skill.

`when_to_use` Optional triggering guidance for this skill.

`paths` Optional file glob patterns that make this skill relevant.

`path` The directory path containing the skill files.

`manifest` Raw YAML frontmatter parsed from SKILL.md.

Methods

Public methods:

- `Skill$new()`
- `Skill$load()`
- `Skill$get_manifest()`
- `Skill$to_store_record()`
- `Skill$metadata_text()`
- `Skill$matches_paths()`
- `Skill$execute_script()`
- `Skill$list_scripts()`
- `Skill$list_resources()`

- Skill$read_resource()
- Skill$get_asset_path()
- Skill$print()
- Skill$clone()

Method `new()`**:** Create a new Skill object by parsing a SKILL.md file.

Usage:

```
Skill$new(path)
```

Arguments:

`path` Path to the skill directory (containing SKILL.md).

Returns: A new Skill object.

Method `load()`**:** Load the full SKILL.md body content (Level 2).

Usage:

```
Skill$load()
```

Returns: Character string containing the skill instructions.

Method `get_manifest()`**:** Return the raw SKILL.md frontmatter as a list.

Usage:

```
Skill$get_manifest()
```

Returns: Named list of metadata fields.

Method `to_store_record()`**:** Export the skill to a structured record suitable for a skill store.

Usage:

```
Skill$to_store_record(
  include_body = TRUE,
  include_files = TRUE,
  include_assets = FALSE
)
```

Arguments:

`include_body` Include the full SKILL.md body in the record.

`include_files` Include textual source files from the skill folder.

`include_assets` Include a list of asset file names.

Returns: A named list ready for JSON serialization.

Method `metadata_text()`**:** Return concise metadata text used for routing and listing.

Usage:

```
Skill$metadata_text()
```

Returns: Character scalar.

Method `matches_paths()`**:** Check whether the skill's `paths` patterns match any provided file paths.

Usage:

```
Skill$matches_paths(file_paths = character(0), cwd = NULL)
```

Arguments:

`file_paths` Character vector of file paths.

`cwd` Optional working directory used to relativize absolute paths.

Returns: Logical scalar.

Method `execute_script()`**:** Execute an R script from the skill's scripts directory (Level 3). Uses callr for safe, isolated execution.

Usage:

```
Skill$execute_script(script_name, args = list())
```

Arguments:

`script_name` Name of the script file (e.g., "normalize.R").

`args` Named list of arguments to pass to the script.

Returns: The result from the script execution.

Method `list_scripts()`**:** List available scripts in the skill's scripts directory.

Usage:

```
Skill$list_scripts()
```

Returns: Character vector of script file names.

Method `list_resources()`**:** List available reference files in the skill's references directory.

Usage:

```
Skill$list_resources()
```

Returns: Character vector of reference file names.

Method `read_resource()`**:** Read content of a reference file from the references directory.

Usage:

```
Skill$read_resource(resource_name)
```

Arguments:

`resource_name` Name of the reference file.

Returns: Character string containing the resource content.

Method `get_asset_path()`**:** Get the absolute path to an asset in the assets directory.

Usage:

```
Skill$get_asset_path(asset_name)
```

Arguments:

`asset_name` Name of the asset file or directory.

Returns: Absolute path string.

Method `print()`**:** Print a summary of the skill.

Usage:

```
Skill$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`skill_registry` *Skill Registry: Scan and Manage Skills*

Description

SkillRegistry class for discovering, caching, and retrieving skills. Scans directories for SKILL.md files and provides access to skill metadata.

`SkillRegistry` *SkillRegistry Class*

Description

R6 class that manages a collection of skills. Provides methods to:

- Scan directories for SKILL.md files
- Cache skill metadata (Level 1)
- Retrieve skills by name
- Generate prompt sections for LLM context

Methods

Public methods:

- `SkillRegistry$new()`
- `SkillRegistry$scan_skills()`
- `SkillRegistry$refresh()`
- `SkillRegistry$list_roots()`
- `SkillRegistry$get_skill()`
- `SkillRegistry$resolve_skill_name()`
- `SkillRegistry$find_closest_skill_name()`
- `SkillRegistry$has_skill()`
- `SkillRegistry$list_skills()`
- `SkillRegistry$count()`
- `SkillRegistry$find_relevant_skills()`
- `SkillRegistry$generate_prompt_section()`

- SkillRegistry$print()
- SkillRegistry$clone()

Method `new()`**:** Create a new SkillRegistry, optionally scanning a directory.

Usage:

```
SkillRegistry$new(path = NULL, recursive = FALSE)
```

Arguments:

`path` Optional path to scan for skills on creation.

`recursive` Whether to scan subdirectories when `path` is provided.

Returns: A new SkillRegistry object.

Method `scan_skills()`**:** Scan a directory for skill folders containing SKILL.md files.

Usage:

```
SkillRegistry$scan_skills(path, recursive = FALSE, remember = TRUE)
```

Arguments:

`path` Path to the directory to scan.

`recursive` Whether to scan subdirectories. Default FALSE.

`remember` Whether this root should be remembered for `refresh()`.

Returns: The registry object (invisibly), for chaining.

Method `refresh()`**:** Re-scan remembered skill roots so updates on disk become visible.

Usage:

```
SkillRegistry$refresh(clear = TRUE)
```

Arguments:

`clear` If TRUE, clears currently loaded skills before re-scanning.

Returns: The registry object (invisibly).

Method `list_roots()`**:** List skill roots remembered by this registry.

Usage:

```
SkillRegistry$list_roots()
```

Returns: A data frame with root path and recursive flag.

Method `get_skill()`**:** Get a skill by name.

Usage:

```
SkillRegistry$get_skill(name)
```

Arguments:

`name` The name of the skill to retrieve.

Returns: The Skill object, or NULL if not found.

Method `resolve_skill_name()`**:** Resolve a skill name or alias to its canonical name.

Usage:

```
SkillRegistry$resolve_skill_name(name)
```

Arguments:

`name` Skill name or alias.

Returns: Canonical skill name or NULL.

Method `find_closest_skill_name()`**:** Find the closest matching canonical skill name for fuzzy recovery.

Usage:

```
SkillRegistry$find_closest_skill_name(name)
```

Arguments:

`name` Skill name or alias candidate.

Returns: Canonical skill name or NULL.

Method `has_skill()`**:** Check if a skill exists in the registry.

Usage:

```
SkillRegistry$has_skill(name)
```

Arguments:

`name` The name of the skill to check.

Returns: TRUE if the skill exists, FALSE otherwise.

Method `list_skills()`**:** List all registered skills with their names and descriptions.

Usage:

```
SkillRegistry$list_skills()
```

Returns: A data.frame with columns: name, description.

Method `count()`**:** Get the number of registered skills.

Usage:

```
SkillRegistry$count()
```

Returns: Integer count of skills.

Method `find_relevant_skills()`**:** Find relevant skills for a user query and optional file paths.

Usage:

```
SkillRegistry$find_relevant_skills(
  query = NULL,
  file_paths = character(0),
  cwd = NULL,
  limit = 3L
)
```

Arguments:

`query` Optional user query text.

`file_paths` Optional character vector of file paths.

`cwd` Optional working directory for path matching.

`limit` Maximum number of results to return.

Returns: Data frame of matching skills sorted by score.

Method `generate_prompt_section()`**:** Generate a prompt section listing available skills. This can be injected into the system prompt.

Usage:

```
SkillRegistry$generate_prompt_section()
```

Returns: Character string with formatted skill list.

Method `print()`**:** Print a summary of the registry.

Usage:

```
SkillRegistry$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`spec_model` *Specification Layer: Model Interfaces*

Description

Abstract base classes (interfaces) for AI models.

`stdlib_agents` *Standard Agent Library: Built-in Specialist Agents*

Description

Factory functions for creating standard library agents with scoped tools and safety guardrails. These agents form the foundation of the multi-agent orchestration system.

`strategy` *Output Strategy System*

Description

Implements the Strategy pattern for handling different structured output formats from LLMs. This allows the SDK to be extended with new output types (e.g., objects, enums, dataframes) without modifying core logic.

stream_image *Stream Image Generation*

Description

Stream image generation with partial-image previews, useful for showing the user progressive renders before the final image arrives. Currently implemented for OpenAI's Responses API (`partial_images` 0–3); other providers raise an informative error directing callers back to `generate_image()`.

Usage

```
stream_image(
  model,
  prompt,
  callback,
  output_dir = tempdir(),
  partial_images = 2,
  registry = NULL,
  ...
)
```

Arguments

`model`	An `ImageModelV1` object or `provider:model` string.
`prompt`	Prompt describing the desired image.
`callback`	A function called for each partial/final event. The event is a named list with `type` (`"partial"` or `"completed"`), `bytes` (raw vector), `media_type` (e.g. `"image/png"`), `index` (integer for partials), and `done` (logical).
`output_dir`	Directory where the final image is written. Defaults to `tempdir()`.
`partial_images`	Integer 0–3 — how many preview frames to request. Defaults to `2`. Set to `0` to disable previews and only receive the final image via the callback.
`registry`	Optional provider registry.
`...`	Additional arguments passed to the model implementation (e.g. `quality`, `output_format`, `background`, `size`).

Value

A `GenerateImageResult` with the final image.

Examples

```
## Not run:
provider <- create_openai()
model <- provider$image_model("gpt-image-1.5")

stream_image(model, "a glowing nebula", callback = function(event) {
```

```
  if (event$type == "partial") {
    message(sprintf("Preview #%d (%d bytes)", event$index, length(event$bytes)))
  } else {
    message("Final image arrived: ", length(event$bytes), " bytes")
  }
})

## End(Not run)
```

`stream_text` *Stream Text*

Description

Generate text using a language model with streaming output. This function provides a real-time stream of tokens through a callback.

Usage

```
stream_text(
  model = NULL,
  prompt,
  callback = NULL,
  system = NULL,
  temperature = 0.7,
  max_tokens = NULL,
  tools = NULL,
  max_steps = 1,
  max_tool_result_errors = 2,
  require_post_tool_protocol = FALSE,
  sandbox = FALSE,
  skills = NULL,
  session = NULL,
  hooks = NULL,
  registry = NULL,
  renderer = NULL,
  .stream_event_callback = NULL,
  ...
)
```

Arguments

`model`	Either a LanguageModelV1 object, or a string ID like "openai:gpt-4o".
`prompt`	A character string prompt, or a list of messages.
`callback`	A function called for each text chunk: `callback(text, done)`.
`system`	Optional system prompt.
`temperature`	Sampling temperature (0-2). Default 0.7.

`max_tokens` Maximum tokens to generate.

`tools` Optional list of Tool objects for function calling.

`max_steps` Number of model/tool steps in one execution window. Default 1. The runtime treats this as a budget checkpoint, not as a hard task stop.

`max_tool_result_errors`
Historical compatibility option. Tool result errors are recorded as task observations; runtime policy decides whether to continue, finalize, ask the user, or block.

`require_post_tool_protocol`
Logical. If TRUE, after any tool results are returned the model must either make another tool call or wrap its final answer in a `<final_answer>...</final_answer>` block. This is enabled automatically for text-based tool fallback.

`sandbox` Logical. If TRUE, enables R-native programmatic sandbox mode. See `generate_text` for details. Default FALSE.

`skills` Optional path to skills directory, or a SkillRegistry object.

`session` Optional ChatSession object for shared state.

`hooks` Optional HookHandler object.

`registry` Optional ProviderRegistry to use.

`renderer` Optional Renderer for agent output. Defaults to the cli terminal backend (`create_stream_renderer()`); pass any Renderer-conforming object (e.g. from a web UI, `create_capture_renderer()`, or `create_null_renderer()`) to render agent output elsewhere.

`.stream_event_callback`
Internal callback for typed stream events.

`...` Additional arguments passed to the model.

Value

A GenerateResult object (accumulated from the stream).

Examples

```
if (interactive()) {
  model <- create_openai()$language_model("gpt-4o")
  stream_text(model, "Tell me a story", callback = function(text, done) {
    if (!done) cat(text)
  })
}
```

sub_session_query *Run a Bounded Child Session*

Description

Creates a scoped child ChatSession for a focused task. The child uses an environment whose parent is the parent session environment, so writes stay in the child scope. Only a compact result summary and trace are written back to the parent context state.

Usage

```
sub_session_query(
  session,
  task,
  context_handles = NULL,
  max_turns = 3L,
  budget = NULL,
  timeout = NULL
)
```

Arguments

session	Parent ChatSession or SharedSession.
task	Focused child task.
context_handles	Optional context handle IDs to summarize for the child.
max_turns	Maximum child generation/tool turns.
budget	Optional budget metadata recorded in the result.
timeout	Optional timeout in seconds. Currently recorded as metadata.

Value

A compact sub-session result list.

Telemetry *Telemetry Class*

Description

R6 class for logging events in a structured format (JSON).

Public fields

`trace_id` Current trace ID for the session.

`events` Collected telemetry events for in-memory inspection.

`emit` Logical; when TRUE, events are printed as JSON lines.

`pricing_table` Pricing for common models (USD per 1M tokens).

Methods

Public methods:

- `Telemetry$new()`
- `Telemetry$log_event()`
- `Telemetry$get_events()`
- `Telemetry$clear_events()`
- `Telemetry$as_hooks()`
- `Telemetry$calculate_cost()`
- `Telemetry$clone()`

Method `new()`**:** Initialize Telemetry

Usage:

```
Telemetry$new(trace_id = NULL, emit = TRUE)
```

Arguments:

`trace_id` Optional trace ID. If NULL, generates a random one.

`emit` Logical; when TRUE, events are printed as JSON lines.

Method `log_event()`**:** Log an event

Usage:

```
Telemetry$log_event(type, ...)
```

Arguments:

`type` Event type (e.g., "generation_start", "tool_call").

`...` Additional fields to log.

Method `get_events()`**:** Return collected telemetry events.

Usage:

```
Telemetry$get_events()
```

Returns: A list of event payloads.

Method `clear_events()`**:** Clear collected telemetry events.

Usage:

```
Telemetry$clear_events()
```

Returns: Invisibly returns self.

Method `as_hooks()`**:** Create hooks for telemetry

Usage:

```
Telemetry$as_hooks()
```

Returns: A HookHandler object pre-configured with telemetry logs.

Method `calculate_cost()`**:** Calculate estimated cost for a generation result

Usage:

```
Telemetry$calculate_cost(result, model_id = NULL)
```

Arguments:

`result` The GenerateResult object.

`model_id` Optional model ID string. if NULL, tries to guess from context (not reliable yet, passing in log_event might be better).

Returns: Estimated cost in USD, or NULL if unknown.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`tool` *Create a Tool*

Description

Factory function to create a Tool object. This is the recommended way to define tools for LLM function calling.

Usage

```
tool(
  name,
  description,
  parameters = NULL,
  execute = NULL,
  layer = "llm",
  meta = NULL
)
```

Arguments

`name`	Unique tool name (used by LLM to call the tool).
`description`	Description of the tool's purpose. Be descriptive to help the LLM understand when to use this tool.
`parameters`	A z_schema object (z_object/z_any/etc), a named list, a character vector, or NULL. When NULL, the schema is inferred from the execute function signature (if possible) and defaults to flexible types.

execute	An R function that implements the tool logic. It can accept a single list argument (args), or standard named parameters. List-style functions receive a single list argument containing parameters.
layer	Tool layer: "llm" (loaded into context) or "computer" (executed via bash/filesystem). Default is "llm". Computer layer tools are not loaded into context but executed via bash.
meta	Optional metadata associated with the tool (e.g., `list(cache_control = list(type = "ephemeral"))`).

Value

A Tool object.

Examples

```
if (interactive()) {
  # Define a weather tool
  get_weather <- tool(
    name = "get_weather",
    description = "Get the current weather for a location",
    parameters = z_object(
      location = z_string(description = "The city name, e.g., 'Beijing'"),
      unit = z_enum(c("celsius", "fahrenheit"), description = "Temperature unit")
    ),
    execute = function(args) {
      # In real usage, call a weather API here
      paste("Weather in", args$location, "is 22 degrees", args$unit)
    }
  )

  # Use with generate_text
  result <- generate_text(
    model = "openai:gpt-4o",
    prompt = "What's the weather in Tokyo?",
    tools = list(get_weather)
  )
}
```

Tool *Tool Class*

Description

R6 class representing a callable tool for LLM function calling. A Tool connects an LLM's tool call request to an R function.

Public fields

`name` The unique name of the tool.

`description` A description of what the tool does.

`parameters` A z_object schema defining the tool's parameters.

`layer` Tool layer: "llm" (loaded into context) or "computer" (executed via bash/filesystem).

`meta` Optional metadata for the tool (e.g., caching configuration).

Methods

Public methods:

- `Tool$new()`
- `Tool$to_api_format()`
- `Tool$run()`
- `Tool$print()`
- `Tool$clone()`

Method `new()`**:** Initialize a Tool.

Usage:

```
Tool$new(name, description, parameters, execute, layer = "llm", meta = NULL)
```

Arguments:

`name` Unique tool name (used by LLM to call the tool).

`description` Description of the tool's purpose.

`parameters` A z_object schema defining expected parameters.

`execute` An R function that implements the tool logic.

`layer` Tool layer: "llm" or "computer" (default: "llm").

`meta` Optional metadata list (e.g., cache_control).

Method `to_api_format()`**:** Convert tool to API format.

Usage:

```
Tool$to_api_format(provider = "openai")
```

Arguments:

`provider` Provider name ("openai" or "anthropic"). Default "openai".

Returns: A list in the format expected by the API.

Method `run()`**:** Execute the tool with given arguments.

Usage:

```
Tool$run(args, envir = NULL)
```

Arguments:

`args` A list or named list of arguments.

`envir` Optional environment in which to evaluate the tool function. When provided, the environment is passed as `.envir` in the args list, allowing the execute function to access and modify session variables.

Returns: The result of executing the tool function.

Method `print()`**:** Print method for Tool.

Usage:

```
Tool$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`update_renviron` *Update .Renviron with new values*

Description

Updates or appends environment variables to the .Renviron file.

Usage

```
update_renviron(updates, path = ".Renviron")
```

Arguments

`updates`	A named list of key-value pairs to update.
`path`	Path to .Renviron file (default: project root)

Value

Invisible TRUE if successful

`utils_capture` *Capture R Console Output*

Description

Internal helpers to capture printed output, messages, and warnings from evaluated R expressions so tool execution can be rendered cleanly in the console UI.

utils_http *Utilities: HTTP and Retry Logic*

Description

Provides standardized HTTP request handling with exponential backoff retry.

Implements multi-layer defense strategy for handling API responses:

- Empty response body handling (returns instead of parse error)
- JSON parsing with repair fallback
- SSE stream error recovery
- Graceful degradation on malformed data

utils_ids *Utilities: Stable IDs*

Description

Generic helpers for generating stable, content-derived IDs.

utils_json *JSON Utilities*

Description

Provides robust utilities for parsing potentially truncated or malformed JSON strings, commonly encountered in streaming LLM outputs.

A robust utility that uses a finite state machine to close open brackets, braces, and quotes to make a truncated JSON string valid for parsing.

Usage

```
fix_json(json_str)
```

Arguments

json_str	A potentially truncated JSON string.

Value

A repaired JSON string.

Examples

```
fix_json('{"name": "Gene...')
fix_json('[1, 2, {"a":')
```

utils_middleware *Utilities: Middleware System*

Description

Provides middleware functionality to wrap and enhance language models. Middleware can transform parameters and wrap generate/stream operations.

utils_registry *Utilities: Provider Registry*

Description

A registry for managing AI model providers. Supports the `provider:model` syntax for accessing models.

variable_registry *Variable Registry*

Description

Provides a mechanism to protect specific variables from being accidentally modified or duplicated by the Agent within the sandbox environment.

walk_ast *Walk an Abstract Syntax Tree*

Description

Recursively traverse an R expression and apply a visitor function to each node.

Usage

```
walk_ast(expr, visitor)
```

Arguments

`expr`	An R expression, call, or primitive type.
`visitor`	A function taking a node as argument.

wrap_language_model *Wrap Language Model with Middleware*

Description

Wraps a LanguageModelV1 with one or more middleware instances. Middleware is applied in order: first middleware transforms first, last middleware wraps closest to the model.

Usage

```
wrap_language_model(model, middleware, model_id = NULL, provider_id = NULL)
```

Arguments

model	A LanguageModelV1 object.
middleware	A single Middleware object or a list of Middleware objects.
model_id	Optional custom model ID.
provider_id	Optional custom provider ID.

Value

A new LanguageModelV1 object with middleware applied.

z_any *Create Any Schema*

Description

Create a JSON Schema that accepts any JSON value.

Usage

```
z_any(description = NULL, nullable = TRUE, default = NULL)
```

Arguments

description	Optional description of the field.
nullable	If TRUE, allows null values.
default	Optional default value.

Value

A list representing JSON Schema for any value.

Examples

```
z_any(description = "Flexible input")
```

z_array *Create Array Schema*

Description

Create a JSON Schema for array type.

Usage

```
z_array(
  items,
  description = NULL,
  nullable = FALSE,
  default = NULL,
  min_items = NULL,
  max_items = NULL
)
```

Arguments

items	Schema for array items (created by z_* functions).
description	Optional description of the field.
nullable	If TRUE, allows null values.
default	Optional default value.
min_items	Optional minimum number of items.
max_items	Optional maximum number of items.

Value

A list representing JSON Schema for array.

Examples

```
z_array(z_string(), description = "List of names")
```

z_boolean *Create Boolean Schema*

Description

Create a JSON Schema for boolean type.

Usage

```
z_boolean(description = NULL, nullable = FALSE, default = NULL)
```

Arguments

description	Optional description of the field.
nullable	If TRUE, allows null values.
default	Optional default value.

Value

A list representing JSON Schema for boolean.

Examples

```
z_boolean(description = "Whether to include details")
```

---

z_dataframe *Create Dataframe Schema*

Description

Create a schema that represents a dataframe (or list of row objects). This is an R-specific convenience function that generates a JSON Schema for an array of objects. The LLM will be instructed to output data in a format that can be easily converted to an R dataframe using `dplyr::bind_rows()` or `do.call(rbind, lapply(..., as.data.frame))`.

Create a schema that represents a dataframe (or list of row objects). This is an R-specific convenience function that generates a JSON Schema for an array of objects.

Usage

```
z_dataframe(
  ...,
  .description = NULL,
  .nullable = FALSE,
  .default = NULL,
  .min_rows = NULL,
  .max_rows = NULL
)
```

Arguments

...	Named arguments where names are column names and values are z_schema objects representing the column types.
.description	Optional description of the dataframe.
.nullable	If TRUE, allows null values.
.default	Optional default value.
.min_rows	Optional minimum number of rows.
.max_rows	Optional maximum number of rows.

Value

A z_schema object representing an array of objects.

A z_schema object representing an array of objects.

Examples

```
# Define a schema for a dataframe of genes
gene_schema <- z_dataframe(
  gene_name = z_string(description = "Name of the gene"),
  expression = z_number(description = "Expression level"),
  significant = z_boolean(description = "Is statistically significant")
)

# Use with generate_object
# result <- generate_object(model, "Extract gene data...", gene_schema)
# df <- dplyr::bind_rows(result$object)
```

---

z_describe *Describe Schema*

Description

Add a description to a z_schema object (pipe-friendly).

Usage

```
z_describe(schema, description)
```

Arguments

schema	A z_schema object.
description	The description string.

Value

The modified z_schema object.

z_empty_object *Create Empty Object Schema*

Description

Create a JSON Schema for an empty object `{}`.

Usage

```
z_empty_object(description = NULL)
```

Arguments

`description`	Optional description.

Value

A z_schema object.

z_enum *Create Enum Schema*

Description

Create a JSON Schema for string enum type.

Usage

```
z_enum(values, description = NULL, nullable = FALSE, default = NULL)
```

Arguments

`values`	Character vector of allowed values.
`description`	Optional description of the field.
`nullable`	If TRUE, allows null values.
`default`	Optional default value.

Value

A list representing JSON Schema for enum.

Examples

```
z_enum(c("celsius", "fahrenheit"), description = "Temperature unit")
```

z_integer *Create Integer Schema*

Description

Create a JSON Schema for integer type.

Usage

```
z_integer(
  description = NULL,
  nullable = FALSE,
  default = NULL,
  minimum = NULL,
  maximum = NULL
)
```

Arguments

description	Optional description of the field.
nullable	If TRUE, allows null values.
default	Optional default value.
minimum	Optional minimum value.
maximum	Optional maximum value.

Value

A list representing JSON Schema for integer.

Examples

```
z_integer(description = "Number of items", minimum = 0)
```

z_number *Create Number Schema*

Description

Create a JSON Schema for number (floating point) type.

Usage

```
z_number(
  description = NULL,
  nullable = FALSE,
  default = NULL,
  minimum = NULL,
  maximum = NULL
)
```

Arguments

description	Optional description of the field.
nullable	If TRUE, allows null values.
default	Optional default value.
minimum	Optional minimum value.
maximum	Optional maximum value.

Value

A list representing JSON Schema for number.

Examples

```
z_number(description = "Temperature value", minimum = -100, maximum = 100)
```

---

z_object *Create Object Schema*

Description

Create a JSON Schema for object type. This is the primary schema builder for defining tool parameters.

Usage

```
z_object(
  ...,
  .description = NULL,
  .required = NULL,
  .additional_properties = FALSE
)
```

Arguments

`...`	Named arguments where names are property names and values are z_schema objects created by z_* functions.
`.description`	Optional description of the object.
`.required`	Character vector of required field names. If NULL (default), all fields are considered required.
`.additional_properties`	Whether to allow additional properties. Default FALSE.

Value

A list representing JSON Schema for object.

Examples

```
z_object(
  location = z_string(description = "City name, e.g., Beijing"),
  unit = z_enum(c("celsius", "fahrenheit"), description = "Temperature unit")
)
```

---

z_string *Create String Schema*

Description

Create a JSON Schema for string type.

Usage

```
z_string(
  description = NULL,
  nullable = FALSE,
  default = NULL,
  min_length = NULL,
  max_length = NULL
)
```

Arguments

`description`	Optional description of the field.
`nullable`	If TRUE, allows null values.
`default`	Optional default value.
`min_length`	Optional minimum string length.
`max_length`	Optional maximum string length.

Value

A list representing JSON Schema for string.

Examples

```
z_string(description = "The city name")
```

Index