

Package ‘hal’

October 6, 2026

Title Agentic Coding Assistants

Version 0.1.4

Description A native R interface to agentic coding assistants. Talks directly to local command line interfaces ('GitHub Copilot', 'Anthropic' 'Claude Code') over standard input/output and to a local 'vscode.lm' bridge inside 'Positron', providing multi-model chat, tool calling, and in-session R evaluation. Supports a single stateful `hal()` session as well as disposable `hal_ask()` / `hal_do()` pipeline verbs, plot capture so models can see graphics, verified data transforms, and user-defined tools.

License MIT + file LICENSE

URL <https://arclite-red.github.io/hal/>,
<https://github.com/ArcLite-Red/hal>

BugReports <https://github.com/ArcLite-Red/hal/issues>

Depends R (>= 4.1)

Imports cli, curl, jsonlite, processx, R6, rlang (>= 1.1.0)

Suggests covr, dplyr, ellmer, ggplot2, knitr, openxlsx2, rmarkdown, rstudioapi, styler, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Mark Dippold [aut, cre]

Maintainer Mark Dippold <arclitered@gmail.com>

Repository CRAN

Date/Publication 2026-10-06 16:20:02 UTC

Contents

hal	2
HalChat	4
HalClient	7
hal_ask	13
hal_available	14
hal_bridge_status	15
hal_chat	16
hal_client	18
hal_config	19
hal_configure	19
hal_do	22
hal_excel	23
hal_history	24
hal_install_bridge	25
hal_models	26
hal_model_info	27
hal_plan	28
hal_quota	28
hal_register_package_tools	29
hal_register_tool	30
hal_register_tools	30
hal_register_tool_specs	31
hal_reset	32
hal_response	33
hal_setup	34
hal_status	35
hal_tool	36
hal_tools	37
hal_tool_call	37
hal_turn	38
hal_usage	39
Index	40

hal	<i>Chat with GitHub Copilot</i>
-----	---------------------------------

Description

The main entry point for conversational interaction with Copilot models. Maintains a persistent session across calls. The SDK agent has built-in tools for file operations, code search, and shell commands.

Usage

```
hal(..., model = NULL, use_env = NULL, reset = FALSE, inspect = FALSE)
```

Arguments

<code>...</code>	Character strings, concatenated as the user message.
<code>model</code>	Model identifier. If NULL, uses the session/configured default.
<code>use_env</code>	Logical or NULL; if TRUE, snapshots the caller's environment and injects it into the prompt so the model naturally uses <code>eval_r</code> to work with your R objects. If NULL (default), auto-detects: scans the prompt for R identifiers and injects the snapshot only when the prompt mentions an object that exists in the caller's environment. When auto-detect fires, a <code>cli::cli_alert_info()</code> announces the matched names (suppress with <code>hal_configure(session_quiet = TRUE)</code>). Set FALSE to disable.
<code>reset</code>	Logical; if TRUE, starts a fresh session before prompting.
<code>inspect</code>	Logical; if TRUE, returns session info instead of prompting.

Value

The assistant's text response (invisibly when displayed).

Errors in scripts

In an interactive session a transport failure prints an alert and returns `invisible(NULL)`. In non-interactive contexts (scripts, R Markdown) it aborts with a classed condition instead, so programmatic callers can handle hal failures specifically: `tryCatch(hal("..."), hal_transport_error = function(e) ...)`. All hal conditions inherit from `hal_error` / `hal_warning`; `hal_do()` signals `hal_do_error` / `hal_do_warning`, and `hal_ask()` signals `hal_ask_warning`.

Examples

```
## Not run:
# Simple conversation
hal("What is R?")
hal("Can you explain more about data frames?")

# Environment access (auto-detected when objects exist)
df <- mtcars
hal("What's the mean mpg?") # auto-detects df, injects env snapshot

# Reset and start fresh
hal("Hello", reset = TRUE)

# Inspect session
hal(inspect = TRUE)

## End(Not run)
```

HalChat

GitHub Copilot Chat Session

Description

High-level chat interface to GitHub Copilot models. Manages conversation history, tool registration, and streaming. API surface mirrors `ellmer::Chat` for familiarity.

Uses [HalClient](#) for JSON-RPC transport to the Copilot SDK CLI, which handles model-specific format translation internally – no proxy translation bugs.

The ACP server maintains conversation history server-side. Turns are also tracked locally so you can inspect them with `$get_turns()`.

Value

An [R6](#) object of class `HalChat`. Methods return values as documented per method; construct with `HalChat$new()`.

Methods

Public methods:

- [HalChat\\$new\(\)](#)
- [HalChat\\$chat\(\)](#)
- [HalChat\\$register_tool\(\)](#)
- [HalChat\\$register_tools\(\)](#)
- [HalChat\\$get_turns\(\)](#)
- [HalChat\\$last_response\(\)](#)
- [HalChat\\$last_turn\(\)](#)
- [HalChat\\$last_tool_calls\(\)](#)
- [HalChat\\$switch_model\(\)](#)
- [HalChat\\$set_mode\(\)](#)
- [HalChat\\$cancel\(\)](#)
- [HalChat\\$get_model\(\)](#)
- [HalChat\\$get_client\(\)](#)
- [HalChat\\$clone\(\)](#)

`HalChat$new()`: Create a new Copilot chat session.

Usage:

```
HalChat$new(  
  model = NULL,  
  system_prompt = NULL,  
  client = NULL,  
  echo = NULL,  
  on_text = NULL,  
  on_tool_call = NULL,
```

```

    on_thought = NULL,
    mode = NULL,
    permission_policy = "auto-allow",
    quiet = FALSE
)

```

Arguments:

`model` Model identifier (e.g., "claude-sonnet-5", "gpt-5.2"). If NULL, uses the server default.

`system_prompt` System prompt string.

`client` A [HalClient](#) instance, or NULL to create one.

`echo` Echo mode: "none", "output", or "all".

`on_text` Callback for streaming text chunks: function(chunk).

`on_tool_call` Callback for tool call events: function(tool_call).

`on_thought` Callback for thought chunks: function(chunk).

`mode` Session mode: "agent" (default), "plan", or "autopilot". Applied after the session is created on the first prompt.

`permission_policy` Permission policy for the agent: "auto-allow", "auto-deny", or a custom function. Ignored if `client` is provided.

`quiet` Logical; suppress informational messages (default: FALSE).

`HalChat$chat()`: Send a message and get a response.

Usage:

```
HalChat$chat(..., timeout = NULL)
```

Arguments:

`...` Character strings, concatenated as the user message.

`timeout` Timeout in seconds for the response.

Returns: Assistant's text response (invisibly if `echo != "none"`).

`HalChat$register_tool()`: Register a tool for the model to call.

Usage:

```
HalChat$register_tool(tool)
```

Arguments:

`tool` A tool definition. Can be an `ellmer::ToolDef` or a list with `name`, `description`, `parameters`, and `fun` fields.

`HalChat$register_tools()`: Register multiple tools.

Usage:

```
HalChat$register_tools(tools)
```

Arguments:

`tools` A list of tool definitions.

`HalChat$get_turns()`: Get conversation turns.

Usage:

`HalChat$get_turns(include_system_prompt = FALSE)`

Arguments:

`include_system_prompt` Include the system prompt turn.

Returns: List of turn objects.

`HalChat$last_response()`: Get the full response from the last prompt.

Usage:

`HalChat$last_response()`

Returns: A `hal_response` object, or `NULL` if no responses yet.

`HalChat$last_turn()`: Get the last assistant turn.

Usage:

`HalChat$last_turn()`

Returns: A `hal_turn` object, or `NULL` if no turns yet.

`HalChat$last_tool_calls()`: Get tool calls from the last assistant turn.

Usage:

`HalChat$last_tool_calls()`

Returns: List of `hal_tool_call` objects, or `NULL` if none.

`HalChat$switch_model()`: Switch models mid-session without losing context.

Usage:

`HalChat$switch_model(model)`

Arguments:

`model` Model identifier (e.g., "gpt-4.1", "claude-haiku-4.5").

Returns: Invisibly returns `self`.

`HalChat$set_mode()`: Set the session mode.

Usage:

`HalChat$set_mode(mode = c("agent", "plan", "autopilot"))`

Arguments:

`mode` "agent" (default), "plan", or "autopilot".

Returns: Invisibly returns `self`.

`HalChat$cancel()`: Cancel the current in-flight prompt.

Stops the streaming loop and returns a partial response with `stop_reason = "interrupted"`. For interactive use, `Ctrl+C` (ESC in RStudio) during `$chat()` achieves the same effect automatically. Useful from Shiny observers, callbacks, or a second R session.

Usage:

`HalChat$cancel()`

Returns: Invisibly returns `self`.

HalChat\$get_model(): Get the model identifier.

Usage:

```
HalChat$get_model()
```

Returns: Character string.

HalChat\$get_client(): Get the underlying client.

Usage:

```
HalChat$get_client()
```

Returns: A [HalClient](#) instance.

HalChat\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

[HalClient](#) for the low-level transport layer, [hal_models\(\)](#) for available models.

Examples

```
## Not run:
# Preferred: use hal_chat()
chat <- hal_chat()
chat$chat("Explain the pipe operator in R")

# Or use R6 constructor directly
chat <- HalChat$new(model = "gpt-5.2")
chat$chat("Hello!")

## End(Not run)
```

Description

Manages the Copilot CLI subprocess in ACP (Agent Client Protocol) mode and handles JSON-RPC 2.0 communication over NDJSON stdio. This is the low-level transport layer – most users should use [HalChat](#) instead.

The client spawns the Copilot CLI with `--acp`, which starts it as a JSON-RPC server. Authentication uses ambient Copilot credentials from any editor (VS Code, Positron, JetBrains).

Value

An [R6](#) object of class HalClient. Methods return values as documented per method; construct with HalClient\$new().

Methods**Public methods:**

- [HalClient\\$new\(\)](#)
- [HalClient\\$start\(\)](#)
- [HalClient\\$stop\(\)](#)
- [HalClient\\$register_tools\(\)](#)
- [HalClient\\$handshake\(\)](#)
- [HalClient\\$new_session\(\)](#)
- [HalClient\\$prompt\(\)](#)
- [HalClient\\$request\(\)](#)
- [HalClient\\$notify\(\)](#)
- [HalClient\\$switch_model\(\)](#)
- [HalClient\\$set_mode\(\)](#)
- [HalClient\\$get_session_id\(\)](#)
- [HalClient\\$swap_session\(\)](#)
- [HalClient\\$restore_session\(\)](#)
- [HalClient\\$cancel\(\)](#)
- [HalClient\\$set_ipc\(\)](#)
- [HalClient\\$is_alive\(\)](#)
- [HalClient\\$read_stderr\(\)](#)
- [HalClient\\$clone\(\)](#)

HalClient\$new(): Create a new Copilot SDK client.

Usage:

```
HalClient$new(
  model = NULL,
  cli_path = NULL,
  permission_policy = "auto-allow",
  on_text = NULL,
  on_tool_call = NULL,
  on_thought = NULL,
  quiet = FALSE
)
```

Arguments:

model Model identifier (e.g., "claude-sonnet-5", "gpt-5.2"). Passed as --model to the CLI at startup. If NULL, falls back to the COPILOT_MODEL environment variable, then the server default.

cli_path Path to the Copilot CLI binary. If NULL, searches PATH and common install locations.

permission_policy How to handle tool permission requests: "auto-allow" (default) auto-approves all, "auto-deny" auto-denies, or a function receiving the permission params and returning an option ID.

on_text Callback function receiving each text chunk as it streams. Signature: `function(chunk)`. Called for each `agent_message_chunk`.

on_tool_call Callback function receiving tool call events. Signature: `function(tool_call)`. Called on `tool_call` and `tool_call_update` events with a `hal_tool_call` object.

on_thought Callback function receiving thought chunks. Signature: `function(chunk)`. Called for each `agent_thought_chunk`.

quiet Logical; suppress informational messages during init, handshake, and session creation (default: FALSE).

HalClient\$start(): Start the Copilot CLI subprocess in ACP mode.

Usage:

`HalClient$start()`

Returns: Invisibly returns self.

HalClient\$stop(): Stop the Copilot CLI subprocess.

Usage:

`HalClient$stop()`

HalClient\$register_tools(): Register tools to expose via MCP server.

Tools must be registered before the first `$prompt()` call (before the CLI subprocess starts). The tools are passed via `--additional-mcp-config` at startup. Registering tools after the CLI is running triggers a warning.

Tools are merged into the existing set by name. Call multiple times to accumulate tools from different sources.

Usage:

`HalClient$register_tools(tools)`

Arguments:

tools Named list of tool definitions.

Returns: Invisibly returns self.

HalClient\$handshake(): Perform the ACP initialize handshake.

Sends the initialize request and initialized notification. Called automatically on first use if needed.

Usage:

`HalClient$handshake(timeout = 15)`

Arguments:

timeout Timeout in seconds.

Returns: The initialize result (agent info and capabilities).

HalClient\$new_session(): Create a new ACP session.

Sends session/new to create a session. The session holds conversation state server-side. Automatically performs the handshake if needed.

Custom tools are configured via --additional-mcp-config at CLI startup (not via mcpServers in this call, which is broken per CLI issue #1040).

Usage:

```
HalClient$new_session(cwd = getwd(), timeout = 15)
```

Arguments:

`cwd` Working directory to report to the server.

`timeout` Timeout in seconds.

Returns: The session result (session ID, available models, modes).

HalClient\$prompt(): Send a prompt and collect the streamed response.

Sends session/prompt and reads session/update notifications until the final response arrives. Automatically creates a session if needed.

Usage:

```
HalClient$prompt(text, timeout = NULL)
```

Arguments:

`text` The prompt text.

`timeout` Timeout in seconds.

Returns: A `hal_response` with `text`, `stop_reason`, `tool_calls`, `thoughts`, and `events`.

HalClient\$request(): Send a JSON-RPC request and wait for a response.

Usage:

```
HalClient$request(method, params = list(), timeout = 60)
```

Arguments:

`method` JSON-RPC method name.

`params` Named list of parameters.

`timeout` Timeout in seconds.

Returns: Parsed JSON response result.

HalClient\$notify(): Send a JSON-RPC notification (no response expected).

Usage:

```
HalClient$notify(method, params = list())
```

Arguments:

`method` JSON-RPC method name.

`params` Named list of parameters.

HalClient\$switch_model(): Switch models mid-session.

Changes the active model without losing conversation context. Use [hal_models\(\)](#) to see available model IDs.

Usage:

```
HalClient$switch_model(model, timeout = 10)
```

Arguments:

`model` Model identifier (e.g., "gpt-4.1", "claude-haiku-4.5").

`timeout` Timeout in seconds.

Returns: Invisibly returns self.

`HalClient$set_mode()`: Set the session mode.

Switches between Agent, Plan, and Autopilot modes.

- **Agent:** Default conversational mode.
- **Plan:** Multi-step planning mode with structured output.
- **Autopilot:** Autonomous mode that runs until task completion without user interaction (experimental).

Usage:

```
HalClient$set_mode(mode = c("agent", "plan", "autopilot"), timeout = 10)
```

Arguments:

`mode` "agent", "plan", or "autopilot".

`timeout` Timeout in seconds.

Returns: Invisibly returns self.

`HalClient$get_session_id()`: Get the current session ID.

Usage:

```
HalClient$get_session_id()
```

Returns: Character string, or NULL if no session is active.

`HalClient$swap_session()`: Create a temporary new session, saving the current one.

Used internally by disposable verbs (`hal_ask`) to get history isolation on the same CLI process.

Call `restore_session()` to switch back.

Usage:

```
HalClient$swap_session()
```

Returns: The saved (previous) session ID (invisibly).

`HalClient$restore_session()`: Restore a previously saved session ID.

Usage:

```
HalClient$restore_session(session_id)
```

Arguments:

`session_id` The session ID returned by `swap_session()`.

Returns: Invisibly returns self.

`HalClient$cancel()`: Cancel the current in-flight prompt.

Signals the streaming loop to stop and return a partial response with `stop_reason = "interrupted"`. Safe to call from callbacks, Shiny observers, or a second thread. Does nothing if no prompt is active.

For interactive use, pressing Ctrl+C (ESC in RStudio) during a prompt achieves the same effect automatically.

Usage:

```
HalClient$cancel()
```

Returns: Invisibly returns self.

HalClient\$set_ipc(): Configure IPC for live eval_r execution.

When set, the polling loop checks for eval_r requests from the MCP subprocess and executes them in the user's R session via eval_fn.

Usage:

```
HalClient$set_ipc(ipc_dir, eval_fn = NULL, permission_fn = NULL)
```

Arguments:

ipc_dir Path to the IPC directory for request/response files.

eval_fn Function taking a code string and returning `list(result = "...", error = NULL)` or `list(result = NULL, error = "...")`.

permission_fn Optional handler for permission requests (used by the Claude permission_prompt bridge). Takes the parsed request object and returns the same `list(result, error)` shape as eval_fn.

Returns: Invisibly returns self.

HalClient\$is_alive(): Check if the client subprocess is running.

Usage:

```
HalClient$is_alive()
```

Returns: Logical.

HalClient\$read_stderr(): Read any available stderr output (for debugging).

Usage:

```
HalClient$read_stderr()
```

Returns: Character string.

HalClient\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

[HalChat](#) for the high-level chat interface, [hal_available\(\)](#) to check CLI availability.

Examples

```
## Not run:
# Preferred: use hal_client()
client <- hal_client()
client$handshake()
session <- client$new_session()
client$stop()

## End(Not run)
```

hal_ask

Ask Copilot a Question – Pipe and Standalone Modes

Description

Pipe mode: a pipe-terminal verb that sends a prompt to Copilot augmented with the piped data context. The piped object is described via `str()` and attached to the prompt. Returns `.data` invisibly (pipe passthrough).

Usage

```
hal_ask(.data, ...)
```

Arguments

<code>.data</code>	In pipe mode: an R object (data frame, model, etc.) piped in. In standalone mode: the first prompt string.
<code>...</code>	Character strings forming the prompt.

Details

Standalone mode: called directly with a string prompt to ask a question without data context. Returns the response text invisibly.

Spawns a disposable session – does NOT pollute the main `hal()` conversation.

Value

Pipe mode: invisibly returns `.data` (pipe passthrough). **Standalone:** invisibly returns the response text.

Examples

```
## Not run:
# Pipe mode (data-aware analysis)
mtcars |> hal_ask("what are the key patterns?")
lm(mpg ~ wt, data = mtcars) |> hal_ask("interpret these coefficients")

# Standalone mode (general question)
hal_ask("what does the pipe operator do in R?")
hal_ask("explain", "the difference between <- and =")

## End(Not run)
```

hal_available

Check whether the active backend's transport is ready

Description

Dispatches by backend:

- vscode: bridge port file exists and /version responds.
- copilot: Copilot CLI binary found and --version succeeds.
- claude: Claude CLI binary found and --version succeeds.

Usage

```
hal_available(backend = NULL)
```

Arguments

backend Optional backend id ("vscode", "copilot", "claude"). If NULL, uses the resolved default for this session.

Details

Honors the `hal.backend` option / explicit argument so callers can probe a specific backend without switching the session default.

Value

TRUE if the backend's transport is reachable, FALSE otherwise.

See Also

[HalChat](#), [hal_models\(\)](#), [hal_bridge_status\(\)](#) for richer vscode-backend diagnostics.

Examples

```
hal_available()  
## Not run:  
hal_available(backend = "vscode")  
hal_available(backend = "claude")  
  
## End(Not run)
```

hal_bridge_status	<i>Check whether hal-bridge is installed and running</i>
-------------------	--

Description

Reads the bridge port file, pings the /version endpoint, and warns on version drift between the running bridge and the version this hal release was pinned against.

Usage

```
hal_bridge_status()
```

Value

Invisibly returns TRUE if the bridge is reachable, FALSE otherwise. Always prints a status line as a side effect.

See Also

[hal_install_bridge\(\)](#).

Examples

```
## Not run:  
hal_bridge_status()  
  
## End(Not run)
```

hal_chat	Create a Copilot chat session
----------	-------------------------------

Description

Creates a new [HalChat](#) instance for conversing with GitHub Copilot models. This is the recommended entry point – equivalent to `HalChat$new()` but follows tidyverse naming conventions.

Usage

```
hal_chat(
  model = NULL,
  system_prompt = NULL,
  client = NULL,
  echo = NULL,
  on_text = NULL,
  on_tool_call = NULL,
  on_thought = NULL,
  mode = NULL,
  permission_policy = "auto-allow",
  quiet = FALSE
)
```

Arguments

model	Model identifier (e.g., "claude-sonnet-5", "gpt-5.2"). Use hal_models() to see available models. If NULL, falls back to the <code>COPILOT_MODEL</code> environment variable, then the server default.
system_prompt	System prompt string. Defaults to hal's built-in R assistant prompt. Pass a string to override, or set via <code>hal_configure(system_prompt = "...")</code> .
client	An existing HalClient instance, or NULL to create one automatically.
echo	Echo mode: "none", "output", or "all". Defaults to "output" in interactive sessions, "none" otherwise.
on_text	Callback for streaming text chunks: <code>function(chunk)</code> . Called for each <code>agent_message_chunk</code> as it arrives.
on_tool_call	Callback for tool call events: <code>function(tool_call)</code> . Called with a <code>hal_tool_call</code> object on tool start and completion.
on_thought	Callback for thought chunks: <code>function(chunk)</code> . Called for each <code>agent_thought_chunk</code> as it arrives.
mode	Session mode: "agent" (default), "plan", or "autopilot". Applied after the session is created on the first prompt.
permission_policy	Permission policy for tool calls. One of "auto-allow" (default), "auto-deny", "ask" (interactive prompt; denies in non-interactive sessions; vscode backend

only), or a function receiving `list(backend, tool_name, input)` and returning a string containing "allow" or "deny". The copilot backend additionally accepts functions returning an ACP option id. Ignored if `client` is provided.

`quiet` Logical; suppress informational messages (default FALSE).

Value

A [HalChat](#) object.

See Also

[hal_models\(\)](#), [hal_client\(\)](#), [hal_available\(\)](#)

Examples

```
## Not run:
# Default model
chat <- hal_chat()
chat$chat("Explain the pipe operator in R")

# Specific model
chat <- hal_chat(model = "gpt-5.2")
chat$chat("Hello!")

# With system prompt
chat <- hal_chat(
  model = "claude-haiku-4.5",
  system_prompt = "Reply concisely in one sentence."
)
chat$chat("What is R?")

# Plan mode
chat <- hal_chat(mode = "plan")
chat$chat("Create an analysis plan for this dataset")

# With streaming callbacks
chat <- hal_chat(
  on_text = function(chunk) cat(chunk),
  on_tool_call = function(tc) message("Tool: ", tc$title)
)

# Read-only mode (deny file writes and shell commands)
chat <- hal_chat(permission_policy = "auto-deny")
chat$chat("What files are in R/?")

## End(Not run)
```

hal_client	Create a Copilot ACP client
------------	-----------------------------

Description

Creates a new [HalClient](#) instance for low-level ACP communication. Most users should use [hal_chat\(\)](#) instead.

Usage

```
hal_client(  
    model = NULL,  
    cli_path = NULL,  
    permission_policy = "auto-allow",  
    on_text = NULL,  
    on_tool_call = NULL,  
    on_thought = NULL,  
    quiet = FALSE  
)
```

Arguments

model	Model identifier (e.g., "claude-sonnet-5", "gpt-5.2"). Passed as --model to the CLI at startup. If NULL, falls back to the COPILOT_MODEL environment variable, then the server default.
cli_path	Path to the Copilot CLI binary. If NULL, searches PATH and common install locations.
permission_policy	How to handle tool permission requests: "auto-allow" (default) auto-approves, "auto-deny" auto-denies, "ask" prompts interactively (vscode backend only), or a function receiving permission params. The function returns an ACP option id on copilot, or a string containing "allow"/"deny" on vscode/claude.
on_text	Callback for streaming text chunks: function(chunk).
on_tool_call	Callback for tool call events: function(tool_call).
on_thought	Callback for thought chunks: function(chunk).
quiet	Logical; suppress informational messages (default FALSE).

Value

A [HalClient](#) object.

See Also

[hal_chat\(\)](#), [hal_available\(\)](#)

Examples

```
## Not run:
client <- hal_client()
client$start()
client$handshake()
session <- client$new_session()
client$stop()

## End(Not run)
```

hal_config	<i>Get Current hal Configuration</i>
------------	--------------------------------------

Description

Returns a named list of all active settings.

Usage

```
hal_config()
```

Value

A named list of configuration values.

Examples

```
cfg <- hal_config()
cfg$backend
cfg$eval_timeout
```

hal_configure	<i>Configure hal Options</i>
---------------	------------------------------

Description

Set display, governance, and session defaults. Settings persist for the duration of the R session via `options()`.

Usage

```

hal_configure(
  use_colors = NULL,
  stream = NULL,
  stream_speed = NULL,
  default_model = NULL,
  system_prompt = NULL,
  eval_denylist = NULL,
  credential_action = NULL,
  eval_timeout = NULL,
  prompt_timeout = NULL,
  prompt_timeout_cold = NULL,
  edit_in_place = NULL,
  do_retries = NULL,
  do_on_fail = NULL,
  verify = NULL,
  plot_vision = NULL,
  permission_policy = NULL,
  session_quiet = NULL,
  init_quiet = NULL,
  show_thoughts = NULL,
  backend = NULL,
  quiet = FALSE
)

```

Arguments

<code>use_colors</code>	Logical; enable ANSI color output (default: TRUE).
<code>stream</code>	Logical; enable streaming typewriter output (default: TRUE).
<code>stream_speed</code>	Character; "instant", "fast", "medium", or "slow" (default: "medium").
<code>default_model</code>	Character; default model identifier.
<code>system_prompt</code>	Character; custom system prompt. Use NULL to keep the current prompt. The placeholder {user} is replaced with the current login name.
<code>eval_denylist</code>	Character vector of function names blocked from eval_r execution. Set to FALSE to disable. Default: curated list.
<code>credential_action</code>	Character; action on credential detection: "warn" (default), "redact", or "block".
<code>eval_timeout</code>	Numeric; seconds for eval_r time limit (default: 30).
<code>prompt_timeout</code>	Numeric; seconds before aborting a resumed prompt on the Claude backend (default: 60). Has no effect on Copilot.
<code>prompt_timeout_cold</code>	Numeric; seconds before aborting the first (session-creating) prompt on the Claude backend (default: 180). The cold path pays for Node.js startup, AV scanning, and session provisioning on Windows and can exceed the warm time-out.

edit_in_place	Logical; when TRUE, hal_do() (and hal_excel()) replaces itself in the editor with generated code. When unset, defaults to TRUE where applicable – i.e. when called from a saved IDE source editor buffer – and FALSE from the console. Set explicitly to force either behaviour.
do_retries	Integer; max retry attempts for hal_do() when generated code fails to parse or execute (default: 2). Set to 0 to disable.
do_on_fail	Character; what hal_do() does when generation fails after all retries: "warn" warns and passes .data through unchanged; "abort" raises an error. When unset, defaults to "warn" in interactive sessions and "abort" in non-interactive contexts (scripts, R Markdown, targets), where silently continuing with untransformed data is worse than failing.
verify	Logical; when TRUE (default), hal_do() pipe mode compares input and output data frames and displays a one-line structural report, attaching the full report as attr(result, "hal_verify"). Report-only: never affects retries or values.
plot_vision	Logical; when TRUE (default), plots drawn by eval_r are captured as PNG and returned to the model as images so it can see and iterate on them (vscode backend with hal-bridge >= 0.1.4, and claude backend). The copilot backend is always text-only regardless of this setting (its CLI's image forwarding is unverified). Notes: plots written to file devices the code opens itself (png(), pdf()) are not echoed; one image per eval (the final page); worst-case eval_r time is ~2x eval_timeout (eval + render).
permission_policy	Character or function; how to handle tool permission requests. "auto-allow" (default) auto-approves; "auto-deny" blocks every tool call; "ask" prompts interactively on the vscode backend (denies in non-interactive sessions); a function receives list(backend, tool_name, input) (copilot adds the full ACP params) and returns a decision – a string containing "allow" or "deny" (vscode/claude), or an ACP option id (copilot). Takes effect on next session init (call hal_reset() to apply immediately).
session_quiet	Logical; suppress informational messages from the transport layer (CLI startup, handshake). Takes effect on next session init (call hal_reset() to apply immediately).
init_quiet	Logical; when TRUE, suppress the data transmission notice shown on first session init (default: FALSE).
show_thoughts	Logical; when TRUE, display the model's reasoning/thought process before the response (default: FALSE). Takes effect on next session init (call hal_reset() to apply immediately).
backend	Character; transport backend: "vscode" (default in Positron) talks to the hal-bridge extension over localhost HTTP fronting vscode.lm; "copilot" (default elsewhere) uses the GitHub Copilot CLI in ACP mode; "claude" uses Anthropic's Claude Code CLI via claude -p with session resume. Takes effect on next session init (call hal_reset() to apply immediately).
quiet	Logical; suppress confirmation messages (default: FALSE).

Value

Invisibly returns a list of changes made.

Examples

```
hal_configure(stream = FALSE, do_retries = 1, quiet = TRUE)
hal_config()$stream
# unset again to restore the defaults
options(hal.stream = NULL, hal.do_retries = NULL)
```

hal_do

Code Generation via LLM – Pipe and Standalone Modes

Description

Pipe mode: a mid-pipe transformation verb that generates R code, executes it, and returns the result.

Usage

```
hal_do(.data, ..., .model = NULL, .retries = NULL, .verify = NULL)
```

Arguments

.data	In pipe mode: an R object piped in. In standalone mode: the first prompt string.
...	Character strings describing the transformation/task.
.model	Model to use for code generation (default: session model).
.retries	Integer; max retry attempts when generated code fails to parse or execute (default: 2, configurable via <code>hal.do_retries</code> option). Set to 0 to disable retries. The retry re-prompts the same disposable session with the error message, so the model can self-correct.
.verify	Logical; in pipe mode with a data-frame result, compare the input and output and display a one-line structural report (row/column deltas, class changes, introduced NAs). The full report is attached as <code>attr(result, "hal_verify")</code> . Report-only: it never affects retries or the returned values, but suspicious patterns (output identical to input, 0-row output) raise a classed <code>hal_do_warning</code> . Default TRUE (configurable via the <code>hal.verify</code> option).

Details

Standalone mode: called directly with a string prompt to generate and execute R code in the caller's environment.

Spawns a disposable session – does NOT pollute the main conversation. Governance controls (denylist, credential scanner, timeout) are enforced.

Value

Pipe mode: transformed data. **Standalone:** result of generated code (invisible).

Failure behavior

In an **interactive** session, a failure (after retries) warns and returns `.data` unchanged (pipe) or `invisible(NULL)` (standalone) – forgiving for console exploration. In **non-interactive** contexts (scripts, R Markdown, targets), it aborts instead: a pipeline silently continuing with untransformed data is worse than an error. Override either way with `options(hal.do_on_fail = "warn")` or `"abort"`. All failures signal classed conditions (`hal_do_error` / `hal_do_warning`), so you can `tryCatch(..., hal_do_error = function(e) ...)`.

Examples

```
## Not run:
# Pipe mode
mtcars |> hal_do("keep only cars with mpg > 25")
iris |> hal_do("add a column for petal area") |> head()

# Standalone mode
hal_do("write a function called greet that takes a name")
greet("world")

# Disable retries
mtcars |> hal_do("normalize mpg", .retries = 0)

## End(Not run)
```

hal_excel

Turn an Excel sheet into R: code that reads the data and recreates the formulas

Description

Reads an `.xlsx`, treats columns without formulas as data, and translates each formula column into a tidyverse expression via the active hal backend. Returns a runnable R script – a line that reads the input columns from the workbook plus a `mutate()` pipeline that recreates the formula columns – so you can replace the spreadsheet (and the `hal_excel()` call) with the code.

Usage

```
hal_excel(path, sheet = 1, model = NULL, tolerance = NULL, retries = NULL)

## S3 method for class 'hal_excel_code'
print(x, ...)
```

Arguments

<code>path</code>	Path to an <code>.xlsx</code> workbook.
<code>sheet</code>	Sheet name, or 1-based index (default 1).

model	Optional model id; defaults to the session/configured model.
tolerance	Numeric tolerance for float comparison (default <code>hal.xl_tolerance</code> , or <code>1e-9</code>).
retries	Max self-correction attempts per column on mismatch (default <code>hal.xl_retries</code> , then <code>hal.do_retries</code> , then 2).
x	A <code>hal_excel_code</code> object (the generated R script).
...	Ignored.

Details

Works like `hal_do()` under the hood (disposable worker, code extraction, denylist, timed eval, retry loop), with one addition: each translation is run and checked against the values Excel cached, so only columns that match Excel row-for-row go into the live pipeline; the rest are emitted as commented stubs to review.

Each formula column is assumed to hold one formula filled down the column (a vectorised expression, e.g. `=A2*B2`), so its first formula is translated once. A workbook with no cached values is still translated but reported unverifiable.

Value

A `hal_excel_code` object (a character string of R code) that prints as the generated script. The per-column report is attached as `attr(result, "hal_excel")`. Save it with `writeln(result, "out.R")`.

Examples

```
## Not run:
hal_excel("model.xlsx")           # prints the read + mutate script
code <- hal_excel("model.xlsx")
writeln(code, "model.R")           # or paste it in place of the call
attr(code, "hal_excel")           # per-column verification report

## End(Not run)
```

hal_history	<i>View Conversation History</i>
-------------	----------------------------------

Description

Display or return the conversation turns from the active session.

Usage

```
hal_history(
  role = NULL,
  pattern = NULL,
  format = c("console", "data.frame", "text"),
  n = NULL
)
```

Arguments

role	Filter by role: "user", "assistant", "system", or NULL for all.
pattern	Optional regex to filter turn content.
format	Output format: "console" (print to screen), "data.frame", or "text" (character vector).
n	Maximum number of turns to return (most recent). NULL for all.

Value

Depends on format: invisible NULL for console, a data.frame, or a character vector.

Examples

```
# with no active session this returns an empty data frame
hal_history(format = "data.frame")
## Not run:
hal_history()           # print all turns
hal_history(role = "user", n = 5) # last five user prompts
hal_history(pattern = "ggplot")  # turns mentioning ggplot

## End(Not run)
```

hal_install_bridge	<i>Install the hal-bridge Positron extension</i>
--------------------	--

Description

Installs the hal-bridge VSIX shipped with this hal release into Positron. No download, no GitHub auth, no SHA verification: the bytes installed are the bytes that shipped in inst/extdata/. Reload Positron after installing to activate the bridge.

Usage

```
hal_install_bridge(local_path = NULL, force = FALSE)
```

Arguments

local_path	Optional path to a .vsix to install instead of the bundled one. Useful for testing a dev build of the bridge.
force	If TRUE, pass --force to the Positron CLI so an existing install is replaced.

Details

The vscode backend (hal_configure(backend = "vscode")) requires this extension. Other backends (copilot, claude) do not.

Value

Invisibly returns the installed VSIX path.

See Also

[hal_bridge_status\(\)](#) to check whether it's actually running.

Examples

```
## Not run:
hal_install_bridge()
hal_install_bridge(local_path = "~/Downloads/hal-bridge-dev.vsix")

## End(Not run)
```

hal_models	<i>List available models</i>
------------	------------------------------

Description

Returns a data frame describing the models exposed by the active backend.

Usage

```
hal_models(client = NULL)
```

Arguments

client A [HalClient](#) instance, or NULL to create one (Copilot only).

Details

- **Copilot:** queries the ACP server (session/new) to fetch live model metadata.
- **Claude:** returns the alias selectors (opus/sonnet/haiku), which Anthropic resolves to the latest model of each tier – the Claude CLI exposes no live model list. Pass a concrete id (e.g. "claude-opus-5") to model= if you need to pin a version.
- **vscode:** queries the hal-bridge /models endpoint, which returns whatever models `vscode.lm` exposes to your Copilot session.

Value

A data frame with columns:

- id — model identifier (matches --model flag)
- name — display name
- description — vendor-supplied prose (may be NA)
- provider — "openai", "anthropic", "google", or "unknown"

- family — coarse tier: "light", "standard", "heavy"
- usage — billing tier label (e.g. "1x", "0.33x", "0x")
- multiplier — numeric form of usage
- is_default — TRUE if currently selected as session default
- context_window — token limit when known (NA otherwise)
- release_date — ISO date when known (NA otherwise)

See Also

[hal_model_info\(\)](#) for full per-model detail (incl. raw _meta).

Examples

```
## Not run:
hal_models()
subset(hal_models(), provider == "anthropic")

## End(Not run)
```

hal_model_info	<i>Inspect a single model in detail</i>
----------------	---

Description

Returns the full per-model record, including the raw _meta payload from the ACP server. Useful for debugging or accessing fields that [hal_models\(\)](#) doesn't surface as columns.

Usage

```
hal_model_info(id = NULL, client = NULL)
```

Arguments

id	Model identifier. If NULL, prints available ids and returns them invisibly.
client	A HalClient instance, or NULL to create one.

Value

A list with id, name, description, provider, family, usage, multiplier, is_default, context_window, release_date, and meta (raw _meta list from the server, or NULL for Claude).

Examples

```
## Not run:
ids <- hal_model_info() # print available ids
hal_model_info(ids[[1]]) # inspect the first one

## End(Not run)
```

hal_plan	<i>Locate the current session's plan.md</i>
----------	---

Description

Plan mode (and occasionally Agent mode) asks the model to commit to a structured plan, written to ~/.copilot/session-state/{sessionId}/plan.md. hal_plan() returns the path so you can view or edit it directly.

Usage

hal_plan()

Value

Character path to plan.md, or NULL invisibly if no plan is available.

Examples

```
## Not run:
hal_reset(mode = "plan")
hal("Investigate the Q4 revenue drop in `sales`")
hal_plan()                # path to plan.md
file.edit(hal_plan())     # open in editor
readLines(hal_plan())     # read programmatically

## End(Not run)
```

hal_quota	<i>Claude 5-hour window quota status</i>
-----------	--

Description

Returns the most recent rate_limit_event payload observed on the Claude backend during this session. Only populated after at least one hal(), hal_ask(), or hal_do() call – the event is embedded in the prompt response stream.

Usage

hal_quota()

Value

A hal_quota object (list) with fields type, status, resets_at (POSIXct), overage_status, overage_disabled_reason, backend. Returns NULL invisibly if no rate-limit info has been observed, or if the active backend is Copilot.

Examples

```
## Not run:  
hal("hello")  
hal_quota()  
  
## End(Not run)
```

`hal_register_package_tools`*Register Functions from an R Package as Tools*

Description

Discovers exported functions from a package and registers them as LLM-callable tools.

Usage

```
hal_register_package_tools(pkg, fns = NULL, exclude = NULL, prefix = NULL)
```

Arguments

<code>pkg</code>	Character; package name.
<code>fns</code>	Character vector of function names. If NULL, registers all exported functions.
<code>exclude</code>	Character vector of function names to skip.
<code>prefix</code>	Optional prefix for tool names (e.g., "dplyr_").

Value

Invisibly returns a list of registered tool definitions.

Examples

```
## Not run:  
# expose a few dplyr verbs to the model  
hal_register_package_tools("dplyr",  
  fns = c("filter", "select", "arrange"), prefix = "dplyr_"  
)  
  
## End(Not run)
```

hal_register_tool	<i>Register an R Function as a Copilot Tool</i>
-------------------	---

Description

Shorthand that builds a tool definition and registers it on the active session's chat object.

Usage

```
hal_register_tool(fn, name = NULL, description = NULL, types = NULL)
```

Arguments

fn	The R function to expose as a tool.
name	Tool name (defaults to the symbol name of fn).
description	Description shown to the LLM (when to call this tool).
types	Optional named list mapping argument names to JSON Schema type strings: "string", "number", "integer", "boolean", "array", "object".

Value

Invisibly returns the tool definition.

Examples

```
## Not run:
get_time <- function(tz = "UTC") format(Sys.time(), tz = tz)
hal_register_tool(get_time,
  description = "Current time in a given timezone"
)
hal("what time is it in Tokyo?")

## End(Not run)
```

hal_register_tools	<i>Register Multiple Tools on the Active Session</i>
--------------------	--

Description

Registers a list of tools on the active `hal()` session so the model can call them. Each element may be an `ellmer::ToolDef` (e.g. from `daisy::daisy_tools()`) or a plain `hal` tool list with name / description / fun / parameters. Unlike `hal_register_tool()`, which builds a tool from a bare function's formals, this passes each element through the chat's tool normalizer, so pre-built `ToolDefs` keep their declared argument schema.

Usage

```
hal_register_tools(tools)
```

Arguments

tools A list of tool definitions (ellmer::ToolDef objects or hal tool lists).

Value

Invisibly, the number of tools registered.

See Also

[hal_register_tool\(\)](#), [hal_register_tool_specs\(\)](#)

Examples

```
## Not run:
tools <- list(
  hal_tool(function(x, y) x + y, "add",
    description = "Add two numbers",
    types = list(x = "number", y = "number")
  ),
  hal_tool(function(path) readLines(path), "read_file",
    description = "Read a text file"
  )
)
hal_register_tools(tools)

## End(Not run)
```

hal_register_tool_specs

Register Tool Specs

Description

Registers a list of tool specifications (as returned by e.g. `winston::timelog_tool_specs()`) on the active session.

Usage

```
hal_register_tool_specs(specs)
```

Arguments

specs A list of tool spec lists.

Details

Each spec must be a list with at minimum name, description, and handler (a function). Optional: parameters (JSON Schema).

Value

Invisibly returns the number of tools registered.

Examples

```
## Not run:
specs <- list(
  list(
    name = "row_count",
    description = "Count the rows of a data frame in the global environment",
    handler = function(name) nrow(get(name, envir = globalenv()))
  )
)
hal_register_tool_specs(specs)

## End(Not run)
```

hal_reset

Reset the hal session

Description

Destroys the current session and starts fresh.

Usage

```
hal_reset(mode = NULL)
```

Arguments

mode	Session mode: "agent" (default), "plan", or "autopilot". If NULL, uses the default (agent).
------	---

Value

Invisibly returns NULL, called for its side effect.

Examples

```
## Not run:
hal_reset()           # fresh session on the same backend
hal_reset(mode = "plan") # restart in plan mode (copilot backend)

## End(Not run)
```

hal_response	<i>hal prompt response</i>
--------------	----------------------------

Description

An S3 class representing the full response from a single prompt. Returned internally by [HalClient](#)'s `prompt()` method and used by [HalChat](#).

Usage

```
## S3 method for class 'hal_response'  
print(x, ...)
```

Arguments

x	A hal_response object.
...	Ignored.

Value

The `print()` method returns x invisibly. `hal_response` objects are created internally by the transport clients.

Fields

`text` Character: the full response text.

`stop_reason` Character: why the model stopped (e.g., "end_turn", "interrupted").

`tool_calls` List of [hal_tool_call](#) objects.

`thoughts` Character vector of agent thought text.

`events` List of raw session/update events.

See Also

[hal_turn](#), [hal_tool_call](#)

`hal_setup`*Set up hal for your environment*

Description

Interactive helper that picks the right backend for the host and walks through prerequisites. Two paths:

Usage

```
hal_setup(force = FALSE, quiet = FALSE, backend = NULL)
```

Arguments

<code>force</code>	Logical; re-run setup even if the backend is already ready.
<code>quiet</code>	Logical; suppress progress messages.
<code>backend</code>	Optional backend override ("vscode", "copilot", "claude"). When NULL, picks vscode in Positron and copilot elsewhere.

Details

- **Positron** (recommended): installs the hal-bridge extension from the VSIX bundled with hal, so hal talks to `vscode.lm` directly. No Node.js, no Copilot CLI, no GitHub CLI, no download – the only requirement is that you are signed in to GitHub Copilot inside Positron itself (the account menu in the lower left).
- **Other hosts** (RStudio, VS Code, command-line R): installs the GitHub Copilot CLI (`gh copilot` extension preferred, `npm @github/copilot` fallback).
- **Claude Code**: verifies the `claude` CLI and points at the download and one-time sign-in. hal does not install this one for you: Claude Code's sign-in is an interactive browser flow, and installing it via `npm` yields a shim whose output is unreliable when driven from R.

Detection is via `POSITRON_VERSION`. Pass `backend = "copilot"` to force the Copilot CLI path inside Positron, or `backend = "claude"` to check the Claude Code CLI instead.

Value

Invisibly TRUE if the backend transport is available after setup, FALSE otherwise.

See Also

[hal_available\(\)](#), [hal_install_bridge\(\)](#), [hal_bridge_status\(\)](#), [hal_models\(\)](#)

Examples

```
## Not run:
hal_setup()                # auto: vscode in Positron, copilot elsewhere
hal_setup(backend = "copilot") # force the Copilot CLI path
hal_setup(backend = "claude")  # check the Claude Code CLI

## End(Not run)
```

hal_status

Diagnose your hal installation in one call

Description

Prints a traffic-light report covering everything between you and a working `hal()` call: which backend is selected (and why), whether its transport is reachable, what the active session looks like, and – when something is wrong – the single next step to fix it.

Usage

```
hal_status()
```

Details

This is the first thing to run when hal misbehaves, and the thing to paste into a bug report.

Value

Invisibly, a list with elements `backend`, `backend_source`, `available`, `detail`, `session_active`, `model`, and `next_step` (NULL when everything is ready).

See Also

[hal_setup\(\)](#), [hal_available\(\)](#), [hal_bridge_status\(\)](#), [hal_config\(\)](#)

Examples

```
## Not run:
hal_status()

## End(Not run)
```

hal_tool

Create a Tool Definition from an R Function

Description

Converts an R function into a hal tool definition suitable for `$register_tool()`. Infers parameter types from function formals; override with the `types` argument for non-string parameters.

Usage

```
hal_tool(fn, name = NULL, description = NULL, types = NULL)
```

Arguments

<code>fn</code>	The R function to expose as a tool.
<code>name</code>	Tool name (defaults to the symbol name of <code>fn</code>).
<code>description</code>	Description shown to the LLM (when to call this tool).
<code>types</code>	Optional named list mapping argument names to JSON Schema type strings: "string", "number", "integer", "boolean", "array", "object".

Value

A list with `name`, `description`, `fun`, and `parameters` – ready for `chat$register_tool()`.

Examples

```
my_sum <- function(x, y) x + y
tool <- hal_tool(my_sum, "add_numbers",
  description = "Add two numbers",
  types = list(x = "number", y = "number")
)
str(tool)
## Not run:
# register on the active session so the model can call it
hal_register_tools(list(tool))

## End(Not run)
```

hal_tools	<i>List Registered Tools</i>
-----------	------------------------------

Description

List Registered Tools

Usage

```
hal_tools()
```

Value

A character vector of tool names registered on the active session.

Examples

```
hal_tools() # character(0) when no session is active
```

hal_tool_call	<i>hal tool call</i>
---------------	----------------------

Description

An S3 class representing a single tool invocation by the agent. Tool calls appear inside [hal_turn](#) objects and are visible when printing conversation history.

Usage

```
## S3 method for class 'hal_tool_call'  
format(x, ...)
```

```
## S3 method for class 'hal_tool_call'  
print(x, ...)
```

Arguments

x	A hal_tool_call object.
...	Ignored.

Value

The print() method returns x invisibly; the format() method returns a character scalar. hal_tool_call objects are created internally as tool activity streams in.

Fields

- tool_call_id Character: unique ID for this tool call.
- title Character: human-readable description (e.g., "Viewing DESCRIPTION").
- kind Character: tool kind ("read", "write", "command", etc.).
- status Character: "pending", "completed", or "failed".
- input Named list of input arguments.
- output Character: output text from the tool.

See Also

[hal_turn](#), [hal_response](#)

hal_turn	<i>hal conversation turn</i>
----------	------------------------------

Description

An S3 class representing one turn in a hal conversation. Turns are returned by [hal_history\(\)](#) and tracked internally by [HalChat](#).

Usage

```
## S3 method for class 'hal_turn'
print(x, ...)
```

Arguments

- x A hal_turn object.
- ... Ignored.

Value

The print() method returns x invisibly. hal_turn objects themselves are created internally and returned by [hal_history\(\)](#).

Fields

- role Character: "user", "assistant", or "system".
- content Character: text content of the turn.
- tool_calls List of [hal_tool_call](#) objects, or NULL.
- thoughts Character vector of agent thought chunks, or NULL.

See Also

[hal_history\(\)](#), [hal_response](#), [hal_tool_call](#)

hal_usage

*Session usage summary***Description**

Tallies `hal()` turns by model and, when tier info is available, computes cumulative Copilot billing units. Usage is tracked per session in memory and cleared by `hal_reset()`.

Usage

```
hal_usage(tiers = NULL)
```

Arguments

<code>tiers</code>	Optional named character vector mapping <code>modelId</code> to usage tier (e.g. <code>c("claude-sonnet-5" = "1x")</code>). When <code>NULL</code> (default), uses the cached tier map populated from the last <code>hal_models()</code> call in this session, or fetches it if none is cached and a live CLI is available. Pass a pre-computed map to skip the network hop.
--------------------	---

Value

A `hal_usage` data frame with columns `model`, `turns`, `tier`, `units` (numeric tier × turns, NA if tier unknown). Prints a compact summary including the session total.

Examples

```
## Not run:
hal("hello")
hal("follow-up", model = "gpt-4.1")
hal_usage()

## End(Not run)
```

Index

`format.hal_tool_call (hal_tool_call)`, 37

`hal`, 2

`hal_ask`, 13

`hal_available`, 14

`hal_available()`, 12, 17, 18, 34, 35

`hal_bridge_status`, 15

`hal_bridge_status()`, 14, 26, 34, 35

`hal_chat`, 16

`hal_chat()`, 18

`hal_client`, 18

`hal_client()`, 17

`hal_config`, 19

`hal_config()`, 35

`hal_configure`, 19

`hal_do`, 22

`hal_do()`, 24

`hal_excel`, 23

`hal_history`, 24

`hal_history()`, 38

`hal_install_bridge`, 25

`hal_install_bridge()`, 15, 34

`hal_model_info`, 27

`hal_model_info()`, 27

`hal_models`, 26

`hal_models()`, 7, 10, 14, 16, 17, 27, 34, 39

`hal_plan`, 28

`hal_quota`, 28

`hal_register_package_tools`, 29

`hal_register_tool`, 30

`hal_register_tool()`, 30, 31

`hal_register_tool_specs`, 31

`hal_register_tool_specs()`, 31

`hal_register_tools`, 30

`hal_reset`, 32

`hal_reset()`, 39

`hal_response`, 33, 38

`hal_setup`, 34

`hal_setup()`, 35

`hal_status`, 35

`hal_tool`, 36

`hal_tool_call`, 33, 37, 38

`hal_tools`, 37

`hal_turn`, 33, 37, 38, 38

`hal_usage`, 39

`HalChat`, 4, 7, 12, 14, 16, 17, 33, 38

`HalClient`, 4, 5, 7, 7, 16, 18, 26, 27, 33

`print.hal_excel_code (hal_excel)`, 23

`print.hal_response (hal_response)`, 33

`print.hal_tool_call (hal_tool_call)`, 37

`print.hal_turn (hal_turn)`, 38

R6, 4, 8