

Package ‘asa’

December 23, 2025

Title AI Search Agent for Large-Scale Research Automation

Version 0.1.0

Description Provides an LLM-powered research agent for performing AI search tasks at large scales. Uses a ReAct (Reasoning + Acting) agent pattern with web search capabilities via DuckDuckGo and Wikipedia. Implements DeepAgent-style memory folding for context management. The agent is built on 'LangGraph' and supports multiple LLM backends including 'OpenAI', 'Groq', and 'xAI'.

URL <https://github.com/cjerzak/asa-software>

BugReports <https://github.com/cjerzak/asa-software/issues>

Depends R (>= 4.0.0)

License GPL-3

Encoding UTF-8

LazyData false

Imports reticulate (>= 1.28),
 jsonlite,
 rlang

Suggests testthat (>= 3.0.0),
 knitr,
 rmarkdown,
 future,
 future.apply

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat.edition 3

SystemRequirements Python (>= 3.11), Conda, Tor (optional, for anonymous searching)

Contents

as.data.frame.asa_result	2
asa_agent	3
asa_response	3
asa_result	4
build_backend	4
build_prompt	5

check_backend	6
configure_search	7
configure_search_logging	8
extract_agent_results	9
extract_search_snippets	9
extract_search_tiers	10
extract_urls	11
extract_wikipedia_content	11
get_agent	12
get_tor_ip	12
initialize_agent	13
is_tor_running	15
print.asa_agent	15
print.asa_response	16
print.asa_result	16
process_outputs	17
reset_agent	17
rotate_tor_circuit	18
run_agent	18
run_agent_batch	19
run_task	20
run_task_batch	21
summary.asa_agent	22
summary.asa_response	23
summary.asa_result	23

Index**24****as.data.frame.asa_result***Convert asa_result to Data Frame***Description**

Convert asa_result to Data Frame

Usage

```
## S3 method for class 'asa_result'
as.data.frame(x, ...)
```

Arguments

- x An asa_result object
- ... Additional arguments (ignored)

Value

A single-row data frame

asa_agent*Constructor for asa_agent Objects*

Description

Creates an S3 object representing an initialized ASA search agent.

Usage

```
asa_agent(python_agent, backend, model, config)
```

Arguments

python_agent	The underlying Python agent object
backend	LLM backend name (e.g., "openai", "groq")
model	Model identifier
config	Agent configuration list

Value

An object of class asa_agent

asa_response*Constructor for asa_response Objects*

Description

Creates an S3 object representing an agent response.

Usage

```
asa_response(  
    message,  
    status_code,  
    raw_response,  
    trace,  
    elapsed_time,  
    fold_count,  
    prompt  
)
```

Arguments

message	The final response text
status_code	Status code (200 = success, 100 = error)
raw_response	The full Python response object
trace	Full text trace of agent execution
elapsed_time	Execution time in minutes
fold_count	Number of memory folds performed
prompt	The original prompt

Value

An object of class `asa_response`

`asa_result`

Constructor for asa_result Objects

Description

Creates an S3 object representing the result of a research task.

Usage

```
asa_result(prompt, message, parsed, raw_output, elapsed_time, status)
```

Arguments

<code>prompt</code>	The original prompt
<code>message</code>	The agent's response text
<code>parsed</code>	Parsed output (list or NULL)
<code>raw_output</code>	Full agent trace
<code>elapsed_time</code>	Execution time in minutes
<code>status</code>	Status ("success" or "error")

Value

An object of class `asa_result`

`build_backend`

Build the Python Backend Environment

Description

Creates a conda environment with all required Python dependencies for the asa search agent, including LangChain, LangGraph, and search tools.

Usage

```
build_backend(conda_env = "asa_env", conda = "auto", python_version = "3.13")
```

Arguments

<code>conda_env</code>	Name of the conda environment (default: "asa_env")
<code>conda</code>	Path to conda executable (default: "auto")
<code>python_version</code>	Python version to use (default: "3.13")

Details

This function creates a new conda environment and installs the following Python packages:

- langchain_groq, langchain_community, langchain_openai
- langgraph
- ddgs (DuckDuckGo search)
- selenium, primp (browser automation)
- beautifulsoup4, requests
- fake_headers, httpx
- pysocks, socksio (proxy support)

Value

Invisibly returns NULL; called for side effects.

Examples

```
## Not run:  
# Create the default environment  
build_backend()  
  
# Create with a custom name  
build_backend(conda_env = "my_asa_env")  
  
## End(Not run)
```

build_prompt

Build a Task Prompt from Template

Description

Creates a formatted prompt by substituting variables into a template.

Usage

```
build_prompt(template, ...)
```

Arguments

template	A character string with placeholders in the form {variable_name}
...	Named arguments to substitute into the template

Value

A formatted prompt string

Examples

```
## Not run:
prompt <- build_prompt(
  template = "Find information about {{name}} in {{country}} during {{year}}",
  name = "Marie Curie",
  country = "France",
  year = 1903
)

## End(Not run)
```

<code>check_backend</code>	<i>Check Python Environment Availability</i>
----------------------------	--

Description

Checks if the required Python environment and packages are available.

Usage

```
check_backend(conda_env = "asa_env")
```

Arguments

<code>conda_env</code>	Name of the conda environment to check
------------------------	--

Value

A list with components:

- `available`: Logical, TRUE if environment is ready
- `conda_env`: Name of the environment checked
- `python_version`: Python version if available
- `missing_packages`: Character vector of missing packages (if any)

Examples

```
## Not run:
status <- check_backend()
if (!status$available) {
  build_backend()
}

## End(Not run)
```

configure_search	<i>Configure Python Search Parameters</i>
------------------	---

Description

Sets global configuration values for the Python search module. These values control timeouts, retry behavior, and result limits.

Usage

```
configure_search(  
    max_results = NULL,  
    timeout = NULL,  
    max_retries = NULL,  
    retry_delay = NULL,  
    backoff_multiplier = NULL,  
    captcha_backoff_base = NULL,  
    page_load_wait = NULL,  
    inter_search_delay = NULL,  
    conda_env = "asa_env"  
)
```

Arguments

max_results	Maximum number of search results to return (default: 10)
timeout	HTTP request timeout in seconds (default: 15)
max_retries	Maximum retry attempts on failure (default: 3)
retry_delay	Initial delay between retries in seconds (default: 2)
backoff_multiplier	Multiplier for exponential backoff (default: 1.5)
captcha_backoff_base	Base multiplier for CAPTCHA backoff (default: 3)
page_load_wait	Wait time after page load in seconds (default: 2)
inter_search_delay	Delay between consecutive searches in seconds (default: 0.5)
conda_env	Name of the conda environment (default: "asa_env")

Value

Invisibly returns a list with the current configuration

Examples

```
## Not run:  
# Increase timeout for slow connections  
configure_search(timeout = 30, max_retries = 5)  
  
# Get more results  
configure_search(max_results = 20)
```

```
# Add delay between searches to avoid rate limiting
configure_search(inter_search_delay = 2.0)

## End(Not run)
```

configure_search_logging*Configure Python Search Logging Level***Description**

Sets the logging level for the Python search module. This controls how much diagnostic output is produced during web searches.

Usage

```
configure_search_logging(level = "WARNING", conda_env = "asa_env")
```

Arguments

<code>level</code>	Log level: "DEBUG", "INFO", "WARNING" (default), "ERROR", or "CRITICAL"
<code>conda_env</code>	Name of the conda environment (default: "asa_env")

Details

Log levels from most to least verbose:

- DEBUG: Detailed diagnostic information for debugging
- INFO: General operational information
- WARNING: Indicates something unexpected but not an error (default)
- ERROR: Serious problems that prevented an operation
- CRITICAL: Very serious errors

Value

Invisibly returns the current logging level

Examples

```
## Not run:
# Enable verbose debugging output
configure_search_logging("DEBUG")

# Run a search (will show detailed logs)
result <- run_task("What is the population of Tokyo?", agent = agent)

# Disable verbose output
configure_search_logging("WARNING")

## End(Not run)
```

`extract_agent_results` *Extract Structured Data from Agent Traces*

Description

Parses raw agent output to extract search snippets, Wikipedia content, URLs, JSON data, and search tier information. This is the main function for post-processing agent traces.

Usage

```
extract_agent_results(raw_output)
```

Arguments

<code>raw_output</code>	Raw output string from agent invocation (the trace field from an <code>asa_response</code> object)
-------------------------	--

Value

A list with components:

- `search_snippets`: Character vector of search result content
- `search_urls`: Character vector of URLs from search results
- `wikipedia_snippets`: Character vector of Wikipedia content
- `json_data`: Extracted JSON data as a list (if present)
- `search_tiers`: Character vector of unique search tiers used (e.g., "primp", "selenium", "ddgs", "requests")

Examples

```
## Not run:
response <- run_agent("Who is the president of France?", agent)
extracted <- extract_agent_results(response$trace)
print(extracted$search_snippets)
print(extracted$search_tiers) # Shows which search tier was used

## End(Not run)
```

`extract_search_snippets`

Extract Search Snippets by Source Number

Description

Extracts content from Search tool messages in the agent trace.

Usage

```
extract_search_snippets(text)
```

Arguments

<code>text</code>	Raw agent trace text
-------------------	----------------------

Value

Character vector of search snippets, ordered by source number

Examples

```
## Not run:
snippets <- extract_search_snippets(response$trace)

## End(Not run)
```

`extract_search_tiers` *Extract Search Tier Information*

Description

Extracts which search tier was used from the agent trace. The search module uses a multi-tier fallback system:

- `primp`: Fast HTTP client with browser impersonation (Tier 0)
- `selenium`: Headless browser for JS-rendered content (Tier 1)
- `ddgs`: Standard DDGS Python library (Tier 2)
- `requests`: Raw POST to DuckDuckGo HTML endpoint (Tier 3)

Usage

```
extract_search_tiers(text)
```

Arguments

<code>text</code>	Raw agent trace text
-------------------	----------------------

Value

Character vector of unique tier names encountered (e.g., "primp", "selenium", "ddgs", "requests")

Examples

```
## Not run:
tiers <- extract_search_tiers(response$trace)
print(tiers) # e.g., "primp"

## End(Not run)
```

`extract_urls`

Extract URLs by Source Number

Description

Extracts URLs from Search tool messages in the agent trace.

Usage

```
extract_urls(text)
```

Arguments

text Raw agent trace text

Value

Character vector of URLs, ordered by source number

Examples

```
## Not run:  
urls <- extract_urls(response$trace)  
  
## End(Not run)
```

`extract_wikipedia_content`

Extract Wikipedia Content

Description

Extracts content from Wikipedia tool messages in the agent trace.

Usage

```
extract_wikipedia_content(text)
```

Arguments

text Raw agent trace text

Value

Character vector of Wikipedia snippets

Examples

```
## Not run:
wiki <- extract_wikipedia_content(response$trace)

## End(Not run)
```

get_agent*Get the Current Agent***Description**

Returns the currently initialized agent, or NULL if not initialized.

Usage

```
get_agent()
```

Value

An `asa_agent` object or NULL

Examples

```
## Not run:
agent <- get_agent()
if (is.null(agent)) {
  agent <- initialize_agent()
}

## End(Not run)
```

get_tor_ip*Get External IP via Tor***Description**

Retrieves the external IP address as seen through Tor proxy.

Usage

```
get_tor_ip(proxy = "socks5h://127.0.0.1:9050")
```

Arguments

<code>proxy</code>	Tor proxy URL
--------------------	---------------

Value

IP address string or NA on failure

Examples

```
## Not run:
ip <- get_tor_ip()
message("Current Tor IP: ", ip)

## End(Not run)
```

initialize_agent	<i>Initialize the ASA Search Agent</i>
------------------	--

Description

Initializes the Python environment and creates the LangGraph agent with search tools (Wikipedia, DuckDuckGo). The agent can use multiple LLM backends and supports DeepAgent-style memory folding.

Usage

```
initialize_agent(
  backend = "openai",
  model = "gpt-4.1-mini",
  conda_env = "asa_env",
  proxy = "socks5h://127.0.0.1:9050",
  use_memory_folding = TRUE,
  memory_threshold = 4L,
  memory_keep_recent = 2L,
  rate_limit = 0.2,
  timeout = 120L,
  verbose = TRUE
)
```

Arguments

backend	LLM backend to use. One of: "openai", "groq", "xai", "exo", "openrouter"
model	Model identifier (e.g., "gpt-4.1-mini", "llama-3.3-70b-versatile")
conda_env	Name of the conda environment with Python dependencies
proxy	SOCKS5 proxy URL for Tor (default: "socks5h://127.0.0.1:9050"). Set to NULL to disable proxy.
use_memory_folding	Enable DeepAgent-style memory compression (default: TRUE)
memory_threshold	Number of messages before folding triggers (default: 4)
memory_keep_recent	Number of recent messages to preserve after folding (default: 2)
rate_limit	Requests per second for rate limiting (default: 0.2)
timeout	Request timeout in seconds (default: 120)
verbose	Print status messages (default: TRUE)

Details

The agent is created with two tools:

- Wikipedia: For looking up encyclopedic information
- DuckDuckGo Search: For web searches with a 4-tier fallback system (PRIMP -> Selenium -> DDGS library -> raw requests)

Memory folding (enabled by default) compresses older messages into a summary to manage context length in long conversations, following the DeepAgent paper.

Value

An object of class `asa_agent` containing the initialized agent and configuration.

API Keys

The following environment variables should be set based on your backend:

- OpenAI: OPENAI_API_KEY
- Groq: GROQ_API_KEY
- xAI: XAI_API_KEY
- OpenRouter: OPENROUTER_API_KEY

OpenRouter Models

When using the "openrouter" backend, model names must be in provider/model-name format.
Examples:

- "openai/gpt-4o"
- "anthropic/clause-3-sonnet"
- "google/gemma-2-9b-it:free"
- "meta-llama/llama-3-70b-instruct"

See <https://openrouter.ai/models> for available models.

See Also

[run_agent](#), [run_task](#)

Examples

```
## Not run:
# Initialize with OpenAI
agent <- initialize_agent(
  backend = "openai",
  model = "gpt-4.1-mini"
)

# Initialize with Groq and custom settings
agent <- initialize_agent(
  backend = "groq",
  model = "llama-3.3-70b-versatile",
  use_memory_folding = FALSE,
  proxy = NULL # No Tor proxy
```

```

)
# Initialize with OpenRouter (access to 100+ models)
agent <- initialize_agent(
  backend = "openrouter",
  model = "anthropic/clause-3-sonnet" # Note: provider/model format
)
## End(Not run)

```

is_tor_running *Check if Tor is Running*

Description

Checks if Tor is running and accessible on the default port.

Usage

```
is_tor_running(port = 9050L)
```

Arguments

port	Port number (default: 9050)
------	-----------------------------

Value

Logical indicating if Tor appears to be running

Examples

```

## Not run:
if (!is_tor_running()) {
  message("Start Tor with: brew services start tor")
}

## End(Not run)

```

print.asa_agent *Print Method for asa_agent Objects*

Description

Print Method for asa_agent Objects

Usage

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

Arguments

- x An asa_agent object
- ... Additional arguments (ignored)

Value

Invisibly returns the object

print.asa_response *Print Method for asa_response Objects*

Description

Print Method for asa_response Objects

Usage

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

Arguments

- x An asa_response object
- ... Additional arguments (ignored)

Value

Invisibly returns the object

print.asa_result *Print Method for asa_result Objects*

Description

Print Method for asa_result Objects

Usage

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

Arguments

- x An asa_result object
- ... Additional arguments (ignored)

Value

Invisibly returns the object

process_outputs	<i>Process Multiple Agent Outputs</i>
-----------------	---------------------------------------

Description

Processes a data frame of raw agent outputs, extracting structured data.

Usage

```
process_outputs(df, parallel = FALSE, workers = 10L)
```

Arguments

df	Data frame with a 'raw_output' column containing agent traces
parallel	Use parallel processing
workers	Number of workers

Value

The input data frame with additional extracted columns: search_count, wiki_count, and any JSON fields found

reset_agent	<i>Reset the Agent</i>
-------------	------------------------

Description

Clears the initialized agent state, forcing reinitialization on next use. Also closes any open HTTP clients to prevent resource leaks.

Usage

```
reset_agent()
```

Value

Invisibly returns NULL

`rotate_tor_circuit` *Rotate Tor Circuit*

Description

Requests a new Tor circuit by restarting the Tor service.

Usage

```
rotate_tor_circuit(method = c("brew", "systemctl", "signal"), wait = 12L)
```

Arguments

<code>method</code>	Method to restart: "brew" (macOS), "systemctl" (Linux), or "signal"
<code>wait</code>	Seconds to wait for new circuit (default: 12)

Value

Invisibly returns NULL

Examples

```
## Not run:
rotate_tor_circuit()

## End(Not run)
```

`run_agent` *Run the ASA Agent with a Custom Prompt*

Description

Invokes the search agent with an arbitrary prompt, returning the full agent trace and response. This is the low-level function for running the agent; for structured task execution, use [run_task](#).

Usage

```
run_agent(prompt, agent = NULL, recursion_limit = NULL, verbose = FALSE)
```

Arguments

<code>prompt</code>	The prompt to send to the agent
<code>agent</code>	An <code>asa_agent</code> object from initialize_agent , or NULL to use/create the default agent
<code>recursion_limit</code>	Maximum number of agent steps (default: 100 for memory folding, 20 otherwise)
<code>verbose</code>	Print status messages (default: FALSE)

Value

An object of class `asa_response` containing:

- `message`: The final response text
- `status_code`: 200 for success, 100 for error
- `raw_response`: The full Python response object
- `trace`: Full text trace of agent execution
- `elapsed_time`: Execution time in minutes
- `fold_count`: Number of memory folds (if memory folding enabled)

See Also

[initialize_agent](#), [run_task](#)

Examples

```
## Not run:
# Run with a custom prompt
agent <- initialize_agent()
result <- run_agent(
  prompt = "Who was the 44th president of the United States?",
  agent = agent
)
print(result$message)

## End(Not run)
```

`run_agent_batch`

Run Agent in Batch Mode

Description

Runs the agent on multiple prompts, optionally in parallel.

Usage

```
run_agent_batch(
  prompts,
  agent = NULL,
  parallel = FALSE,
  workers = 4L,
  progress = TRUE
)
```

Arguments

<code>prompts</code>	Character vector of prompts
<code>agent</code>	An <code>asa_agent</code> object
<code>parallel</code>	Use parallel processing (requires <code>future.apply</code> package)
<code>workers</code>	Number of parallel workers (default: 4)
<code>progress</code>	Show progress bar (default: TRUE)

Value

A list of `asa_response` objects

Examples

```
## Not run:
prompts <- c(
  "What is the population of Tokyo?",
  "What is the population of New York?"
)
results <- run_agent_batch(prompts, agent)

## End(Not run)
```

`run_task`

Run a Structured Task with the Agent

Description

Executes a research task using the AI search agent with a structured prompt and returns parsed results.

Usage

```
run_task(prompt, output_format = "text", agent = NULL, verbose = FALSE)
```

Arguments

<code>prompt</code>	The task prompt or question for the agent to research
<code>output_format</code>	Expected output format. One of: "text" (raw response), "json" (parse as JSON), or a character vector of field names to extract
<code>agent</code>	An <code>asa_agent</code> object from initialize_agent , or <code>NULL</code> to use the currently initialized agent
<code>verbose</code>	Print progress messages (default: <code>FALSE</code>)

Details

This function provides a high-level interface for running research tasks. For simple text responses, use `output_format = "text"`. For structured outputs, use `output_format = "json"` or specify field names to extract.

Value

An object of class `asa_result` with components:

- `prompt`: The original prompt
- `message`: The agent's response text
- `parsed`: Parsed output (if `output_format` specified)
- `raw_output`: Full agent trace
- `elapsed_time`: Execution time in minutes
- `status`: "success" or "error"

See Also

[initialize_agent](#), [run_agent](#), [run_task_batch](#)

Examples

```
## Not run:
# Initialize agent first
agent <- initialize_agent(backend = "openai", model = "gpt-4.1-mini")

# Simple text query
result <- run_task(
  prompt = "What is the capital of France?",
  output_format = "text",
  agent = agent
)
print(result$message)

# JSON structured output
result <- run_task(
  prompt = "Find information about Albert Einstein and return JSON with
           fields: birth_year, death_year, nationality, field_of_study",
  output_format = "json",
  agent = agent
)
print(result$parsed)

## End(Not run)
```

run_task_batch

Run Multiple Tasks in Batch

Description

Executes multiple research tasks, optionally in parallel.

Usage

```
run_task_batch(
  prompts,
  output_format = "text",
  agent = NULL,
  parallel = FALSE,
  workers = 4L,
  progress = TRUE
)
```

Arguments

- | | |
|---------------|--|
| prompts | Character vector of task prompts, or a data frame with a 'prompt' column |
| output_format | Expected output format (applies to all tasks) |
| agent | An asa_agent object |

<code>parallel</code>	Use parallel processing
<code>workers</code>	Number of parallel workers
<code>progress</code>	Show progress messages

Value

A list of `asa_result` objects, or if `prompts` was a data frame, the data frame with result columns added

Examples

```
## Not run:
prompts <- c(
  "What is the population of Tokyo?",
  "What is the population of New York?",
  "What is the population of London?"
)
results <- run_task_batch(prompts, agent = agent)

## End(Not run)
```

`summary.asa_agent` *Summary Method for asa_agent Objects*

Description

Summary Method for `asa_agent` Objects

Usage

```
## S3 method for class 'asa_agent'
summary(object, ...)
```

Arguments

<code>object</code>	An <code>asa_agent</code> object
<code>...</code>	Additional arguments (ignored)

Value

Invisibly returns a summary list

summary.asa_response *Summary Method for asa_response Objects*

Description

Summary Method for asa_response Objects

Usage

```
## S3 method for class 'asa_response'  
summary(object, show_trace = FALSE, ...)
```

Arguments

object	An asa_response object
show_trace	Include full trace in output
...	Additional arguments (ignored)

Value

Invisibly returns a summary list

summary.asa_result *Summary Method for asa_result Objects*

Description

Summary Method for asa_result Objects

Usage

```
## S3 method for class 'asa_result'  
summary(object, ...)
```

Arguments

object	An asa_result object
...	Additional arguments (ignored)

Value

Invisibly returns a summary list

Index

as.data.frame.asa_result, 2
asa_agent, 3
asa_response, 3
asa_result, 4

build_backend, 4
build_prompt, 5

check_backend, 6
configure_search, 7
configure_search_logging, 8

extract_agent_results, 9
extract_search_snippets, 9
extract_search_tiers, 10
extract_urls, 11
extract_wikipedia_content, 11

get_agent, 12
get_tor_ip, 12

initialize_agent, 13, 18–21
is_tor_running, 15

print.asa_agent, 15
print.asa_response, 16
print.asa_result, 16
process_outputs, 17

reset_agent, 17
rotate_tor_circuit, 18
run_agent, 14, 18, 21
run_agent_batch, 19
run_task, 14, 18, 19, 20
run_task_batch, 21, 21

summary.asa_agent, 22
summary.asa_response, 23
summary.asa_result, 23