Package ‘hellmer’

March 7, 2025

Title Batch Processing for Chat Models

Version 0.1.0

Description Batch processing framework for 'ellmer' chat model interactions.
Enables sequential and parallel processing of chat completions.
Core capabilities include error handling with backoff, state persistence,
progress tracking, and retry management.
Parallel processing is implemented via the 'future' framework.
Additional features include structured data extraction, tool integration,
timeout handling, verbosity control, and sound notifications.
Includes methods for returning chat texts, chat objects, progress status,
and structured data.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

Imports beepr, cli, future, furrr, parallel, purrr, R.utils, S7, utils

Depends ellmer

URL `https://dylanpieper.github.io/hellmer/`

NeedsCompilation no

Author Dylan Pieper [aut, cre]

Maintainer Dylan Pieper `<dylanpieper@gmail.com>`

Repository CRAN

Date/Publication 2025-03-07 11:40:10 UTC

Contents

batch *Batch class for managing chat processing*

Description

Batch class for managing chat processing

Usage

```
batch(
  prompts = list(),
  responses = list(),
  completed = integer(0),
  state_path = character(0),
  type_spec = NULL,
  echo = character(0),
  input_type = character(0),
  max_retries = integer(0),
  initial_delay = integer(0),
  max_delay = integer(0),
  backoff_factor = integer(0),
  chunk_size = integer(0),
  workers = integer(0),
  plan = character(0),
  state = list()
)
```

Arguments

`prompts`	List of prompts to process
`responses`	List to store responses
`completed`	Integer indicating number of completed prompts
`state_path`	Path to save state file
`type_spec`	Type specification for structured data extraction
`echo`	Level of output to display ("none", "text", "all")

`input_type`	Type of input ("vector" or "list")
`max_retries`	Maximum number of retry attempts
`initial_delay`	Initial delay before first retry
`max_delay`	Maximum delay between retries
`backoff_factor`	Factor to multiply delay by after each retry
`chunk_size`	Size of chunks for parallel processing
`workers`	Number of parallel workers
`plan`	Parallel backend plan
`state`	Internal state tracking

Value

Returns an S7 class object of class "batch" that represents a collection of prompts and their responses from chat models. The object contains all input parameters as properties and provides methods for:

- Extracting text responses via `texts()`
- Accessing full chat objects via `chats()`
- Tracking processing progress via `progress()`
- Extracting structured data via `structured_data()` when a type specification is provided

The batch object manages prompt processing, tracks completion status, and handles retries for failed requests.

Examples

```
# Create a chat processor
chat <- chat_sequential(chat_openai())

# Process a batch of prompts
batch <- chat$batch(list(
  "What is R?",
  "Explain base R versus tidyverse",
  "Explain vectors, lists, and data frames"
))

# Check the progress if interrupted
batch$progress()

# Return the responses as a vector or list
batch$texts()

# Return the chat objects
batch$chats()
```

chats — *Extract chat objects from a batch result*

Description

Extract chat objects from a batch result

Extract chat objects from a batch result

Usage

```
chats(x, ...)
```

Arguments

`x`	A batch object
`...`	Additional arguments passed to methods

Value

A list of chat objects

A list of chat objects

Examples

```
# Create a chat processor
chat <- chat_sequential(chat_openai())

# Process a batch of prompts
batch <- chat$batch(list(
  "What is R?",
  "Explain base R versus tidyverse",
  "Explain vectors, lists, and data frames"
))

# Return the chat objects
batch$chats()
```

chat_future *Process a batch of prompts in parallel*

Description

Processes a batch of chat prompts using parallel workers. Splits prompts into chunks for processing while maintaining state. For sequential processing, use `chat_sequential()`.

Usage

```
chat_future(
  chat_model = NULL,
  workers = parallel::detectCores(),
  plan = "multisession",
  chunk_size = NULL,
  max_chunk_attempts = 3L,
  max_retries = 3L,
  initial_delay = 20,
  max_delay = 60,
  backoff_factor = 2,
  timeout = 60,
  beep = TRUE,
  ...
)
```

Arguments

`chat_model`	ellmer chat model function or object (e.g., `ellmer::chat_claude`)
`workers`	Number of parallel workers to use (default: number of CPU cores)
`plan`	Processing strategy to use: "multisession" for separate R sessions or "multicore" for forked processes (default: "multisession")
`chunk_size`	Number of prompts to process in parallel at a time (default: 10% of the number of prompts)
`max_chunk_attempts`	Maximum number of retry attempts for failed chunks (default: 3L)
`max_retries`	Maximum number of retry attempts per prompt (default: 3L)
`initial_delay`	Initial delay in seconds before first retry (default: 20)
`max_delay`	Maximum delay in seconds between retries (default: 60)
`backoff_factor`	Factor to multiply delay by after each retry (default: 2)
`timeout`	Maximum time in seconds to wait for each prompt response (default: 2)
`beep`	Logical to play a sound on batch completion, interruption, and error (default: TRUE)
`...`	Additional arguments passed to the underlying chat model (e.g., `system_prompt`)

Value

A batch object (S7 class) containing:

- prompts: Original input prompts
- responses: Raw response data for completed prompts
- completed: Number of successfully processed prompts
- state_path: Path where batch state is saved
- type_spec: Type specification used for structured data
- texts: Function to extract text responses
- chats: Function to extract chat objects
- progress: Function to get processing status
- structured_data: Function to extract structured data (if `type_spec` was provided)

Examples

```
# Create a parallel chat processor
chat <- chat_future(chat_openai, system_prompt = "Reply concisely, one sentence")

# Process a batch of prompts in parallel
batch <- chat$batch(list(
  "What is R?",
  "Explain base R versus tidyverse",
  "Explain vectors, lists, and data frames"
))

# Check the progress if interrupted
batch$progress()

# Return the responses as a vector or list
batch$texts()

# Return the chat objects
batch$chats()
```

`chat_sequential` *Process a batch of prompts in sequence*

Description

Processes a batch of chat prompts one at a time in sequential order. Maintains state between runs and can resume interrupted processing. For parallel processing, use `chat_future()`.

Usage

```
chat_sequential(
  chat_model = NULL,
  echo = "none",
  max_retries = 3L,
  initial_delay = 20,
  max_delay = 60,
  backoff_factor = 2,
  timeout = 60,
  beep = TRUE,
  ...
)
```

Arguments

`chat_model`	ellmer chat model function or object (e.g., `ellmer::chat_claude`)
`echo`	Level of output to display: "none" for silent operation, "text" for response text only, or "all" for full interaction (default: "none")
`max_retries`	Maximum number of retry attempts per prompt (default: 3L)
`initial_delay`	Initial delay in seconds before first retry (default: 20)
`max_delay`	Maximum delay in seconds between retries (default: 60)
`backoff_factor`	Factor to multiply delay by after each retry (default: 2)
`timeout`	Maximum time in seconds to wait for each prompt response (default: 60)
`beep`	Logical to play a sound on batch completion, interruption, and error (default: TRUE)
`...`	Additional arguments passed to the underlying chat model (e.g., `system_prompt`)

Value

A batch object (S7 class) containing

- prompts: Original input prompts
- responses: Raw response data for completed prompts
- completed: Number of successfully processed prompts
- state_path: Path where batch state is saved
- type_spec: Type specification used for structured data
- texts: Function to extract text responses
- chats: Function to extract chat objects
- progress: Function to get processing status
- structured_data: Function to extract structured data (if `type_spec` was provided)

Examples

```
# Create a sequential chat processor
chat <- chat_sequential(chat_openai, system_prompt = "Reply concisely, one sentence")

# Process a batch of prompts in sequence
batch <- chat$batch(list(
  "What is R?",
  "Explain base R versus tidyverse",
  "Explain vectors, lists, and data frames"
))

# Check the progress if interrupted
batch$progress()

# Return the responses as a vector or list
batch$texts()

# Return the chat objects
batch$chats()
```

`progress` *Get progress information from a batch result*

Description

Get progress information from a batch result

Usage

```
progress(x, ...)
```

Arguments

`x`	A batch object
`...`	Additional arguments passed to methods

Value

A list containing progress details

Examples

```
# Create a chat processor
chat <- chat_sequential(chat_openai())

# Process a batch of prompts
batch <- chat$batch(list(
  "What is R?",
  "Explain base R versus tidyverse",
  "Explain vectors, lists, and data frames"
))

# Check the progress
batch$progress()
```

`progress.batch` *Extract progress information from a batch*

Description

Extract progress information from a batch

Arguments

`x`	A batch object

Value

A list containing progress details

`structured_data` *Extract structured data from a batch result*

Description

Extract structured data from a batch result

Usage

```
structured_data(x, ...)
```

Arguments

`x`	A batch object
`...`	Additional arguments passed to methods

Value

A list of structured data objects

Examples

```
# Create a chat processor with type specification
book_type <- type_object(
  title = type_string(),
  author = type_string(),
  year = type_integer()
)

# Create chat processor
chat <- chat_sequential(chat_openai())

# Process a batch of prompts with type spec
batch <- chat$batch(list(
  "Return info about 1984 by George Orwell",
  "Return info about Brave New World by Aldous Huxley"
), type_spec = book_type)

# Extract structured data
batch$structured_data()
```

`structured_data.batch` *Extract structured data from a batch*

Description

Extract structured data from a batch

Arguments

`x`	A batch object

Value

List of structured data

`texts` *Extract texts from a batch result*

Description

Extract texts from a batch result

Usage

```
texts(x, ...)
```

Arguments

x	A batch object
...	Additional arguments passed to methods

Value

A character vector or list of text responses

Examples

```
# Create a chat processor
chat <- chat_sequential(chat_openai())

# Process a batch of prompts
batch <- chat$batch(list(
  "What is R?",
  "Explain base R versus tidyverse",
  "Explain vectors, lists, and data frames"
))

# Extract text responses
batch$texts()
```

texts.batch *Extract text responses from a batch*

Description

Extract text responses from a batch

Arguments

x	A batch object
flatten	Logical; whether to flatten structured data into a single string (default: TRUE)

Value

A character vector (if original prompts were supplied as a vector) or a list of response texts (if original prompts were supplied as a list)

Index