Package: hellmer (via r-universe)

March 14, 2025

Title Batch Processing for Chat Models

Version 0.1.1

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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

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

Depends ellmer

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

Config/pak/sysreqs libssl-dev

Repository https://dylanpieper.r-universe.dev

RemoteUrl https://github.com/dylanpieper/hellmer

RemoteRef HEAD

RemoteSha 715d2fe1969f4f4996dc6421031acf3f1f800c82

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,
  judgements = integer(0),
  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

judgements	Number of judgements in a batch_judge() workflow (1 = initial extract + 1 judgement, 2 = initial extract + 2 judgements, etc.)
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() (includes structured data when a type specification is provided)
- Accessing full chat objects via chats()
- Tracking processing progress via progress()

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

Usage

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

Arguments

x	A batch object
...	Additional arguments

Value

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 = 80,
  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: number of prompts / 10)
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: 80)
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 (includes structured data when a type specification is provided)
- chats: Function to extract chat objects
- progress: Function to get processing status

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
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 = 80,
  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: 80)
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 (includes structured data when a type specification is provided)
- chats: Function to extract chat objects
- progress: Function to get processing status

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
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()
```

texts *Extract texts or structured data from a batch result*

Description

Extract texts or structured data 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. If a type specification was provided to the batch, structured data objects will be returned instead.

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()
```

Index