Package 'hellmer'

April 17, 2025

Title Batch Processing for Chat Models

Version 0.1.2

Description Batch processing framework for 'ellmer' chat models.
Provides both sequential and parallel processing of chat interactions
with features including tool calling and structured data extraction.
Enables workflow management through progress tracking and recovery and
automatic retry with backoff. Additional quality-of-life features
include verbosity (or echo) control and sound notifications.
Parallel processing is implemented via the 'future' framework.
Includes methods for retrieving progress status, chat texts, and chat objects.

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, jsonlite, parallel, purrr, 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 2025-04-17 18:40:02 UTC

Contents

`batch` *Batch result class for managing chat processing results*

Description

Batch result class for managing chat processing results

Usage

```
batch(
  prompts = list(),
  responses = list(),
  completed = integer(0),
  state_path = character(0),
  type_spec = NULL,
  judgements = integer(0),
  progress = logical(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),
  beep = logical(0),
  echo = logical(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.)
progress	Whether to show progress bars (default: TRUE)
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
beep	Play sound on completion (default: TRUE)
echo	Whether to echo messages during processing (default: FALSE)
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()
```

`batch.future_chat` *Process a batch of prompts with a parallel chat*

Description

Process a batch of prompts with a parallel chat

Usage

```
batch.future_chat(
  chat_env,
  prompts,
  type_spec = NULL,
  judgements = 0,
  state_path = tempfile("chat_", fileext = ".rds"),
  workers = NULL,
  chunk_size = parallel::detectCores() * 5,
  plan = "multisession",
  max_chunk_attempts = 3L,
  max_retries = 3L,
  initial_delay = 20,
  max_delay = 80,
  backoff_factor = 2,
  beep = TRUE,
  progress = TRUE,
  echo = FALSE,
  ...
)
```

Arguments

`chat_env`	The chat environment from chat_future
`prompts`	List of prompts to process
`type_spec`	Type specification for structured data extraction
`judgements`	Number of judgements for structured data extraction resulting in refined data
`state_path`	Path to save state file
`workers`	Number of parallel workers
`chunk_size`	Number of prompts each worker processes at a time
`plan`	Parallel backend ("multisession" or "multicore")
`max_chunk_attempts`	Maximum retries per failed chunk
`max_retries`	Maximum number of retry attempts for failed requests
`initial_delay`	Initial delay before first retry in seconds
`max_delay`	Maximum delay between retries in seconds

backoff_factor	Factor to multiply delay by after each retry
beep	Whether to play a sound on completion
progress	Whether to show progress bars
echo	Whether to display chat outputs (when progress is FALSE)
...	Additional arguments passed to the chat method

Value

A batch object with the processed results

---

batch.sequential_chat *Process a batch of prompts with a sequential chat*

Description

Process a batch of prompts with a sequential chat

Usage

```
batch.sequential_chat(
  chat_env,
  prompts,
  type_spec = NULL,
  judgements = 0,
  state_path = tempfile("chat_", fileext = ".rds"),
  progress = TRUE,
  max_retries = 3L,
  initial_delay = 20,
  max_delay = 80,
  backoff_factor = 2,
  beep = TRUE,
  echo = FALSE,
  ...
)
```

Arguments

chat_env	The chat environment from chat_sequential
prompts	List of prompts to process
type_spec	Type specification for structured data extraction
judgements	Number of judgements (1 = initial extract + 1 judgement, 2 = initial extract + 2 judgements, etc.)
state_path	Path to save state file
progress	Whether to show progress bars

max_retries	Maximum number of retry attempts for failed requests
initial_delay	Initial delay before first retry in seconds
max_delay	Maximum delay between retries in seconds
backoff_factor	Factor to multiply delay by after each retry
beep	Whether to play a sound on completion
echo	Whether to display chat outputs (when progress is FALSE)
...	Additional arguments passed to the chat method

Value

A batch object with the processed results

---

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, ...)
```

Arguments

`chat_model`	ellmer chat model object or function (e.g., `chat_openai()`)
`...`	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 or structured data
- **chats**: Function to extract chat objects
- **progress**: Function to get processing status
- **batch**: Function to process a batch of prompts

Batch Method

This function provides access to the `batch()` method for parallel processing of prompts. See `?batch.future_chat` for full details of the method and its parameters.

Examples

```
# Create a parallel chat processor with an object
chat <- chat_future(chat_openai(system_prompt = "Reply concisely"))

# Or a function
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"
  ),
  chunk_size = 3
)

# Process batch with echo enabled (when progress is disabled)
batch <- chat$batch(
  list(
    "What is R?",
    "Explain base R versus tidyverse"
  ),
  progress = FALSE,
  echo = TRUE
)

# 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, ...)
```

Arguments

`chat_model`	ellmer chat model object or function (e.g., `chat_openai()`)
`...`	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 or structured data
- **chats**: Function to extract chat objects
- **progress**: Function to get processing status
- **batch**: Function to process a batch of prompts

Batch Method

This function provides access to the `batch()` method for sequential processing of prompts. See `?batch.sequential_chat` for full details of the method and its parameters.

Examples

```
# Create a sequential chat processor with an object
chat <- chat_sequential(chat_openai(system_prompt = "Reply concisely"))

# Or a function
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"
  ),
  max_retries = 3L,
  initial_delay = 20,
  beep = TRUE
)

# Process batch with echo enabled (when progress is disabled)
batch <- chat$batch(
  list(
    "What is R?",
    "Explain base R versus tidyverse"
  ),
  progress = FALSE,
  echo = TRUE
)

# 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