Mentat

Release 1.0.18

Multiple Authors

Apr 30, 2024

CONTENTS:

CHAPTER

ONE

USER GUIDE

1.1 Getting Started

1.1.1 Installation

It is recommended you install this package in a virtualenv.

```
# Python 3.10 or higher is required
python3 -m venv .venv
source .venv/bin/activate
```

Install the package from PyPI:

```
python -m pip install mentat
```

From github:

```
python -m pip install git+https://github.com/AbanteAI/mentat.git
```

Or via brew:

```
brew install mentat
```

Extras

If you want to use the whisper transcription on an OS besides macOS or Windows you will need to install PortAudio. For instance, on Ubuntu:

```
sudo apt-get install libportaudio2
```

1.1.2 Basic Usage

Start mentat with a list of files you want mentat to be able to read and edit and ask for what you want:

```
mentat file1 file2 file3
```

You can add and remove files from context in a session with the `/include` and `/exclude` commands. For more on context see *Context*. For a list of all commands see *Commands* or enter `/help` in a mentat session.

1.2 Context

Context refers to portions of your code/files which are sent to the LLM along with your question/task. An LLM is only as good as its context. Therefore we provide a number of ways to manually manage and inspect context as well as a few tools to attempt to automatically select a context.

Files can be manually added to context as a command line argument when starting mentat or with the `/include` command during a session. Files can be removed from context with the `/exclude` command. Mentat always puts all included files into the system message sent to the LLM so you probably don't want to start mentat with `mentat .`. If you do want mentat to intelligently select the context from your prompt you should run `mentat -a` and mentat will build its own context. For more see *Auto Context*.

You can specify line ranges to add only a subset of a file to context by adding the starting line (inclusive) and ending line (exclusive) to the path. For example `/include README.md:1-5,10-20` would add lines 1, 2, 3 and 4 and 10th to 19th lines to the LLMs context.

You can see the conversation exactly as the LLM sees it by running `/viewer`. This command opens the transcript in a web browser. If you click a message from the LLM you will see the conversation as the LLM sees it. You can see past conversations by using the arrow keys.

1.2.1 Auto Context

If you enable auto context either by starting mentat with the `-a` flag or by setting `auto_context_tokens` to a positive number during a session then on every request mentat will put `auto_context_tokens` (defaults to 5000 if `-a` is used with no argument) many tokens to your system prompt code message. Those tokens are chosen via embeddings.

A similar thing one can do is use the `/search` command to get related snippets of code and then add them to context from the search interface.

Manually included files are still included when you enable auto context so you shouldn't run `mentat . -a`.

1.3 Commands

In addition to the standard chat interface there are a number of commands available to do things such as modify or observe the context, run shell commands, use vision and voice, and more. You can get help information for them in a session by running the `/help` command.

1.3.1 /agent

Toggle agent mode. In agent mode Mentat will automatically make changes and run commands.

1.3.2 /clear

Clear the current message history and auto included code features from context.

1.3.3 /commit [commit message]

Commit all unstaged and staged changes to git.

1.3.4 /config <setting> [value]

Show or set a config option's value.

1.3.5 /exclude <path|glob pattern> ...

Remove files from context.

1.3.6 /help [command] ...

Show information on available commands.

1.3.7 /include <path|glob pattern> ...

Add files to context.

1.3.8 /load [context file path]

Load context from a file.

1.3.9 /save [context file path]

Save context to a file.

1.3.10 /redo

Redo a change that was previously undone with /undo.

1.3.11 /run <command> [args] ...

Run a shell command and put its output in context. For example, run a python script with arguments:

```
run python my_script.py arg1 arg2
```

This command is very useful to avoid copy pasting. Instead of pasting the test output into mentat and asking mentat to fix it, simply run the test in mentat.

1.3.12 /screenshot [path|url]

Open a url or local file in a web browser with Selenium, take a screenshot and put it into the model's context. The model is automatically changed to gpt-4-vision-preview if an openai model that doesn't support vision is currently set.

1.3.13 /search <query>

Search files in context semantically with embeddings. Results can be added to context directly from the search interface.

1.3.14 /talk

Start voice to text. Uses whisper via the openai api.

1.3.15 /undo

Undo the last change made by Mentat.

1.3.16 /undo-all

Undo all changes made by Mentat.

1.3.17 /viewer

Open a webpage showing the conversation so far. Model messages can be clicked to show the conversation from the model's perspective. There are buttons to share the conversation or give feedback.

1.3.18 /amend

Used to amend a previous user request. Works by resetting context to the state it was at the last request and prefills user input with the last request. Does not undo any edits.

1.4 Configuration

Mentat has a number of customizable settings. They can be changed globally by setting in `~/.mentat/.mentat_config.json` or on a per project level by setting in `.mentat_config.json`. They can also be set as a command line flag, see *Command Line Arguments*, note underscores become dashes on the cli. Finally they can be changed mid-session with the `/config` command.

The following is a partial list of customizable settings.

1.4.1 Settings

model

The model used for making edits. We recommend sticking to gpt-4-1106-preview. For changing to non-openai models see *Alternative Models*.

maximum_context

The maximum number of tokens to put into context. When using an openai model this defaults to the model's maximum context. Otherwise it defaults to 4096.

auto_context_tokens

When this is set to a positive integer that many tokens of additional context are selected with an embeddings system and put into context. For more see *Auto Context*.

theme

Currently only `dark` and `light` are supported.

temperature

The temperature used by the edit generating model. This defaults to 0.2.

no_parser_prompt

When this is set to true the model isn't given a system prompt describing how to make edits. This should only be set for fine tuned models.

embedding_model

The model used for making embeddings.

file_exclude_glob_list

List of glob patterns to exclude files from being read/edited by Mentat. These take effect whenever you provide Mentat with a directory as an argument. Mentat will add all files in the directory that are not in your `.gitignore` and do not match these glob patterns. Glob patterns are interpreted from the git root location. If you wanted to exclude all files ending in `.py`, the pattern to use would be `**/*.py` rather than `*.py`. Here is an example that would exclude all hidden directories and files:

```
{
    "file-exclude-glob-list": ["**/.*, **/.*/**"]
}
```

parser

Mentat is able to edit files by parsing a specific format that the model is told to follow. We are always working to improve the format we use, and have multiple formats available. Although we expect the default format to perform the best, you can test out other formats using the configuration.

```
{
    "parser": "block"
}
```

Available formats:

- block
- replacement
- unified-diff

1.5 Alternative Models

1.5.1 Anthropic's Claude 3

To use Anthropic models, provide the `ANTHROPIC_API_KEY` environment variable instead of `OPENAI_API_KEY`, and set the model *claude-3-opus-20240229* in the `.mentat_config.json` file:

```
# in ~/.mentat/.env
ANTHROPIC_API_KEY=sk-**********

# In ~/.mentat/.mentat_config.json
{ "model": "claude-3-opus-20240229" }
```

1.5.2 OpenAI models on Azure

To use the Azure API, provide the `AZURE_OPENAI_ENDPOINT` (`https://<your-instance-name>.openai.azure.com/`) and `AZURE_OPENAI_KEY` environment variables instead of `OPENAI_API_KEY`.

In addition, Mentat uses the `gpt-4-1106-preview` model by default. When using Azure, you will have to set the model as described in *Configuration* to the name you gave your Azure model.

> **Warning:** Due to changes in the OpenAI Python SDK, you can no longer use `OPENAI_API_BASE` to access the Azure API with Mentat.

1.5.3 Using Other Models

Mentat uses the OpenAI SDK to retrieve chat completions. This means that setting the *OPENAI_API_BASE* environment variable is enough to use any model that has the same response schema as OpenAI. To use models with different response schemas, we recommend setting up a litellm proxy as described here and pointing *OPENAI_API_BASE* to the proxy. For example:

```
pip install 'litellm[proxy]'
litellm --model huggingface/bigcode/starcoder --drop_params
# Should see: Uvicorn running on http://0.0.0.0:8000
```

```
# In ~/.mentat/.env
OPENAI_API_BASE=http://localhost:8000
# or
export OPENAI_API_BASE=http://localhost:8000
```

1.5.4 Local Models

This works the same as in the previous section but you must install ollama first. Replace mixtral with whichever model you want to use.

```
ollama pull mixtral
ollama serve
# Should see: listening on 127.0.0.1:11434
```

Next run the litellm proxy. In another terminal run:

```
pip install 'litellm[proxy]'
litellm --model ollama/mixtral --api_base http://localhost:11434 --drop_params
# Should see: Uvicorn running on http://0.0.0.0:8000
```

Finally set the OPENAI_API_BASE in the terminal before running mentat.

Note: When using a litellm proxy, the model set in Mentat's config will not affect the model being run. To change the model, rerun the litellm proxy with a different model. As mentat thinks it is talking to gpt-4-1106-preview you may want to set `maximum_context`.

Warning: Be sure to include the –drop_params argument when running the litellm proxy! Mentat uses some arguments (such as response_format) that may not be available in alternative models.

1.6 Command Line Arguments

CHAPTER

TWO

PYTHON SDK GUIDE

You can import Mentat in your python programs and use it programmatically. For an example see the *usage* section. For full specs of the python module see the *module* section.

2.1 Python SDK Usage

The Python SDK allows you to use mentat in your python programs. If you have mentat installed you can use it like this:

```
from mentat import Mentat
client = Mentat(paths=['README.md'])

client.startup()
client.call_mentat_auto_accept("Please fix the typos in the Readme.")
client.shutdown()
```

All the same options are available as in the command line interface. You can use commands and change configuration by passing in a mentat.Config object.

2.2 Python Client

2.2.1 mentat.python_client.client module

CHAPTER

THREE

DEVELOPER GUIDE

3.1 mentat package

3.1.1 Subpackages

mentat.command package

Subpackages

mentat.command.commands package

Submodules

mentat.command.commands.agent module

mentat.command.commands.clear module

mentat.command.commands.commit module

mentat.command.commands.config module

mentat.command.commands.context module

mentat.command.commands.exclude module

mentat.command.commands.help module

mentat.command.commands.include module

mentat.command.commands.redo module

mentat.command.commands.run module

mentat.command.commands.screenshot module

Submodules

mentat.prompts.prompts module

Module contents

mentat.python_client package

Submodules

mentat.python_client.client module

Module contents

mentat.terminal package

Submodules

mentat.terminal.client module

mentat.terminal.loading module

mentat.terminal.output module

mentat.terminal.prompt_completer module

mentat.terminal.prompt_session module

Module contents

mentat.vision package

Submodules

mentat.vision.vision_manager module

Module contents

3.1.2 Submodules

3.1.3 mentat.agent_handler module

3.1.4 mentat.app_conf module

3.1.5 mentat.auto_completer module

3.1.6 mentat.broadcast module

3.1.7 mentat.code_context module

3.1.8 mentat.code_edit_feedback module

3.1.9 mentat.code_feature module

3.1.10 mentat.code_file_manager module

3.1.11 mentat.config module

3.1.12 mentat.conversation module

3.1.13 mentat.diff_context module

3.1.14 mentat.edit_history module

3.1.15 mentat.errors module

3.1.16 mentat.git_handler module

3.1.17 mentat.include_files module

3.1.18 mentat.interval module

3.1.19 mentat.llm_api_handler module

3.1.20 mentat.logging_config module

3.1.21 mentat.sentry module

3.1.22 mentat.session module

3.1.23 mentat.session_context module

3.1.24 mentat.session_input module

3.1.25 mentat.session_stream module

3.1.26 mentat.streaming_printer module

3.1.27 mentat.transcripts module

CHAPTER

FOUR

CHANGELOG

In this changelog focus on user facing highlights and stick to the format. This information will be used to motivate users to upgrade or after upgrading to inform them of features that might otherwise not be very discoverable.

4.1 1.0.18

- Bug fixes and dependency updates

4.2 1.0.17

- Fix requirement conflict for numpy

4.3 1.0.16

- Always use the latest patch-level versions of Spice and Ragdaemon

4.4 1.0.15

- Improved auto-context selection and generation
- Improved token counting, cost tracking, and message conversion for Anthropic models
- Fixed vision inputs with Anthropic models
- Switched default model to gpt-4-turbo

4.5 1.0.14

- Fixed bugs relating to VSCode extension
- Updated dependencies

4.6 1.0.13

- Claude 3 support
- All Anthropic models can now be used without requiring a LiteLLM proxy

4.7 1.0.12

- Added helpful message when no api key found
- Fixed errors relating to embedding models

4.8 1.0.11

- Added /save and /load command to save and load context selections
- Changed format to fit Anthropic models
- Other bug fixes

4.9 1.0.10

- Mentat is now a full terminal app which displays the context and running cost in the sidebar.
- Mentat now has a python sdk. Try *from mentat import Mentat* to get started. See the docs for more details.
- New openai models added to the model list.

4.10 1.0.9

- Adds */amend* command: clear last message and prefill with last prompt.
- Experimental feature revisor. Turn on with *–revisor* flag. Attempts to fix edits that fail to conform to parser format.
- Switch to ChromaDB for embeddings.

4.11 1.0.8

- Auto context now only grows so the model won't forget earlier read files.
- Faster embeddings for search and auto context.
- Share button added to */viewer*.
- Improved documentation for non OpenAI models.

4.12 1.0.7

- */search* command now has UI to add found files to context.
- Feedback button added to */viewer*.
- Command and file autocompletion.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

M