UIUC CS 410 Text Information Systems class of Fall 2023

Final Project report

by Nina Stawski (NET ID: ninas2)

The purpose of the application is to help fix some simple issues with the pull requests: typos, basic code composition, inconsistent style, by adding actionable comments to the pull request review.

Setup instructions and video report

Setup instructions can be found in README.md
Video report can be found at https://youtu.be/IpcNBNLbRoo

The general application flow

Here is how the application works.

1. Connect to GitHub and fetch all open pull requests in the specific project
2. For each pull request, fetch the diff
3. For each diff, send it to ChatGPT with a prompt
4. Receive and clean the response
5. Post it as a GitHub PR comment

Approach

I considered two approaches to building such a bot: through webhooks and as a standalone application that is querying the API.

While webhooks would be a more scalable approach and would allow for pushing the events to the application rather than polling over a regular period of time, it raised a number of security concerns and would require a lot more development effort. Since a webhook application would be running online open to the internet, it would require validating the input and making sure it came from GitHub, making sure it can run continuously, would require setting up a web server, etc. etc. After some deliberation I opted to write an application that can be ran on a localhost with personal tokens provided by the person running it – a lot easier to control and debug.

Running the application locally requires generating personal tokens for GitHub and for OpenAI, so it requires some effort on the grader’s part. However, it is more secure for the specific purpose since no service/endpoint needs to be maintained for it to work.

Implementation notes

The most of the time spent on the project was in the prompt engineering. Getting a reliable response from GPT is not a trivial task. Here are the options I have tried:

1. Adding a new file with text
2. Adding a new file with random code (language not mentioned in the prompt)
3. Adding a new file with the specific code (language hardcoded in the prompt)
4. Adding a change into the file that already exists in the repository.

I arrived at the prompt where I would mention the language specifically and provide notes as to how I want the comments and the output look. Still, there are glitches in the response – additional symbols, forgotten braces, incorrect line numbers, which require re-running the script multiple times. I was not able to achieve complete consistency with the response.

The main issue with the ChatGPT API was that it has unpredictability in the response, and despite all the instructions, the final response would often come malformed and unsuitable for posting to GitHub. To mitigate this, I added 10 retry attempts on errors and some output cleaning. In some cases, it still might be necessary to restart the script altogether.

Even when I thought I arrived at a stable solution, the next day the results returned from the OpenAI API could be completely different (for example, lacking specific code snippets besides the language in the query asking to provide those).

Additional limitations were on the implementation. For example, PyGithub doesn't let you publish comments into a draft review, and all the comments created are visible right away. This provides some limitations: the comments cannot be edited before they are visible, they cannot be all posted as a part of the whole review etc.

Final evaluation

Have you completed what you have planned? Have you got the expected outcome? If not, discuss why.

I was able to build the bot that checks for the pull requests and gets the feedback on these pull requests. When ran enough times, some of the comments the bot provides are helpful (especially ones regarding typos etc.). Most of the comments are also placed in the correct location. I can say that I achieved my objective.

However, it wasn't possible to make the script completely reliable, so it still doesn't produce great comments places exactly where they need to be and formatter perfectly every time. To remediate that, I had to implement retries and still, sometimes you need to restart the Python script altogether. This is due to flexible nature of the generative language models.

I originally thought this bot would be flexible and useful for multiple languages, however, specifying the programming language in the prompt proved to yield better results. In the future, this can be remediated by an additional call to OpenAI to determine the language of the repo / script, and then choosing from one of the predefined prompts.

The project can still be improved in multiple ways:

- Increase the reliability of the responses by tweaking the prompt and providing additional context to the chatbot (other existing files in a project, style guides, etc.)
- Run the script as a webhook so it could react at new pull request actions automatically
- Additional checks and error handling in case the comment location is out of bounds
- Improved logging (not just print() statements)