Part 2: Designing a Interface Between ChatGPT and CODA

Implementation

I built on my part 1 implementation by adding an interpreter function to translate JSON into executable code and modified my prompt to have the model output JSON commands instead of a shell script.

Prompt

The prompt was carefully engineered over many iterations. I decided to give the model an outline of what JSON commands to use as well as careful explanations of each field in the JSON because it kept outputting inconsistent JSON commands. I also gave the model specific instructions for creating and updating files as without these instructions it would modify random python files in the working directory. Finally, I included a copy of the model's previous code if the prompt was a follow up response as the model would rewrite code without the context of the first prompt without this addition.

Interpreter

The interpreter executed the JSON commands given by the model. Each JSON model had the following form:

```
"action": "create|update|delete|execute",
"target": "filename" to create, delete, or containing code to execute,
"content": "script content here if creating",
"before": "code to be modified from previous response if updating",
"after": "modified code segment if updating",
"language": "python3",
"arguments": ["arg1", "arg2"] if executing
```

Difficulties

I had a bunch of difficulty with prompt engineering! First, the model would overwrite my main.py whenever a follow up question was asked instead of modifying its own code. Next, the model would completely forget the context of the first prompt when replying to the second. It would also misunderstand actions and execute the incorrect actions for different tasks. Sometimes, it would try to overwrite input files instead of writing code. I also had to change the follow up prompt a bit by replacing the words update your code with update chatgpt_code.py as the model just could not remember the filename it had chosen in the previous response. I had many more difficulties like this, resulting in lots of time fine tuning my prompts!

AI

AI was surprisingly unhelpful with helping fine tune my prompt, but Clarity did write my interpreter and main function. It also gave me a base prompt to work with and helped brainstorm and implement security measures.

Security Measures

- The validate_target function has most of my security checks

    - Checks that the file created/updated is called chatgpt_code.py (in the prompt, I tell the model to only modify that file so that no other files are overwritten or modified, and all code is kept in one place)
    - Checks that the file extension is in the allowed FILE_EXTENSIONS defined at the top of the file to ensure no harmful files are created
    - Checks that the file created/modified is in the working directory so the model doesn't access any files it shouldn't outside the current directory
    - Checks that the file to be modified is not greater than a given MAX_FILE_SIZE
- For the delete command, ensures any directories in IMPORTANT_DIRECTORIES given at the top are not deleted

Timing

Unsurprisingly, the shell script implementation was much faster than the JSON command implementation – the average time for the shell script to be created and executed was 2.7792 seconds in comparison to the average 8.3644 seconds it took for the JSON implementation to be created and executed. As the JSON implementation has to go through the model, the interpreter, and many checks before execution, it takes much longer compared to the simple method of the shell script – script from model and then straight to execution.

Proof

All proof is attached in the repository