CSCE 689: Special Topics in Programming Large Language Models (LLMs)
Homework 1
Student: Mu-Ruei Tseng
UIN: 133007868

Repo location: /home/u133007868/CSCE-689-PROGRAMMING-LLMs/homeworks/hw1
For this assignment, I provided three models: gpt-4o-mini, gemini-1.5-flash, and the local llama3.1. To use the model, one has to add the corresponding api key in the .env file. For example:

``` .env
OPENAI_KEY=<your_openai_api_key>
GEMINI_KEY=<your_gemini_api_key>
```

The pricing for each api is provided below:

gpt-4o-mini:
- Input pricing: $0.15 per 1 million tokens
- Output pricing: $0.6 per 1 million tokens

gemini-1.5-flash: For prompts up to 128k tokens:
- Input pricing: $0.075 per 1 million tokens
- Output pricing: $0.30 per 1 million tokens

For gemini-1.5-flash, we follow this pricing structure as generating x.bin uses approximately 350 tokens, and generating x.diff requires around 1300 tokens, both well within the 128k token limit.

Get started:
pip install openai
pip install -q -U google-generativeai

Important arguments for the python file:
- model: Specify the model to be used. ("openai", "gemini", "local_llama3"). Default: openai.
- max_limit: Maximum number of attempts to generate the x.bin file. Default: 10.
- deterministic: Use deterministic mode for the model. Default: False

---

Generating x.bin:
To ask the model to generate a potential x.bin that would cause the mock_vp.c code to raise an error, I passed the entire mock_vp.c code as a string to the LLM and instructed it to produce input that might trigger a vulnerability. I also specified that the vulnerability could include buffer overflows, memory leaks, misuse of pointers, input validation issues, and more, ensuring that the search wasn't hardcoded to look for a specific vulnerability.

To capture the model's response effectively, I imposed constraints on the output format, as the LLM can generate various types of responses, such as explanations or additional characters wrapping around the answers. Without restricting the response format, extracting the relevant information becomes

challenging. To address this, I instructed the LLM to return only the response that adheres to the specific JSON format I provided. The JSON object I used is as follows:

```
{
   'func_a': '<array of strings>',
   'func_b': '<int>'
}
```

However, depending on the model selected, the response may still include brief information wrapping the answer, as shown in the example below.

```python
{
   'func_a': '<array of strings>',
   'func_b': '<int>'
}
```

Therefore, I have a post-processing step to use regular expressions to find the pattern r'({.*})'. This is designed to match text that starts with { and ends with }, which is the structure of a JSON object. This allows me to better constraint and utilize the model output. I extracted the information from the json object and processed them to be in the desired format for x.bin.

Although the constraints I provided significantly improve the likelihood of the model generating the correct failure case for the C code, I also implemented a validation mechanism after generating the x.bin using the testing command line. I continue prompting the LLM until the response passes the validation check, with the set maximum iterations.

The total time required, number of tokens used (including prompt tokens and completion tokens) will be provided at the end of generating the x.bin. Here is an example of the console output:

```
Using the model: openai
==========Generating x.bin==========
Attempt 1
Successfully generated the x.bin that triggers the vulnerability.
Token count: 294
Prompt Tokens: 268
Completion Tokens: 26
Time taken (s):  1.53
```

Generating x.diff:
Generating x.diff involves multiple steps. In the preliminary experiment, I asked the model to directly output a diff file that could be used to update the .c file. I provided the model with the mock_vp.c code and also the harness code. However, the model's response often failed to correctly build the patch. For example:

error: patch failed: mock_vp.c:22

error: mock_vp.c: patch does not apply
Patching failed using: /home/u133007868/CSCE-689-PROGRAMMING-LLMs/homeworks/hw1/x.diff

To address the issue, I asked the model to output the modified C code that fixes the potential vulnerabilities. I also emphasized that it should only correct these vulnerabilities while preserving the original functionality of the code. Unlike generating x.bin, where the goal is input generation, I wanted the LLM to directly produce C code, which often includes many " and \n characters. This frequently causes issues when parsing into JSON if not properly escaped. Therefore, I explicitly requested the C code in the following format:

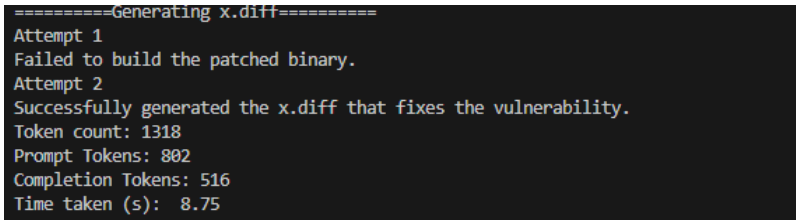
```c
<modified c code>
```

After extracting the modified C code from the response, I save it to a temporary file and use Python to execute the command line for generating the diff file (diff -u {src_path} ./tmp.c) and capture the output. A post-processing step is applied to remove the first two lines of the diff output, as we want the diff file to be applied to the same source file.

--- a/mock_vp.c
+++ b/mock_vp.c

After the post-processing, the x.diff file can be saved. The same loop mechanism is applied to ensure valid output.

The total time required, number of tokens used (including prompt tokens and completion tokens) will also be provided at the end of generating the x.diff. Here is an example of the console output:



From the exemple, it shows that the generated answer fails to pass the validation during the first attempt and is asked to regenerate the answer. The second iteration then generated a correct diff file that passed the test.

Bonus:

I also implemented support for using the local Llama 3.1 model. To run this task, set the model as local_llama3. I used the same prompt for generating x.bin and x.diff. However, with the local LLM, the inference time is significantly longer, and the likelihood of failure is higher compared to the other two models, which typically generate the correct answer in just one attempt. One might want to increase the max iterations to 20.