Harvard CS 2420: Computing at Scale (Fall 2025) **(Due: 2025/10/18 at 11:59PM EST)**

Problem Set 3 (PSet 3): Language Model Assisted Development

Instructor: H.T. Kung

Team: Name-1, Name-2, Name-3, Name-4

Please include the full names of all team members in your submission.

Introduction

Assignment Objectives

The objectives of this assignment are to:

- Provide hands-on experience with various language models (LMs).
- Explore strengths and weaknesses of LMs in debugging and programming workflows.

Important note: this assignment is intended to be an open-ended exercise. If a model suggests or implements an objectively incorrect solution, we encourage documenting and analyzing the outcome! We hope this will be a useful exercise to increase your familiarity and comfort with LMs in a technical setting.

Python Notebook and Colab

The code will be provided as an interactive Python notebook on Canvas, along with this handout. You can run it on Google Colab or locally (e.g., with Jupyter or VSCode) if you wish.

Language Model Access

For this assignment, we recommend using the Harvard AI Sandbox, which provides access to a variety of models, including OpenAI's GPT-4, Anthropic's Claude, and Meta's LLama. We encourage exploring multiple model families (e.g., LLama vs Claude), generations (e.g., GPT-4 vs GPT-5) and sizes (e.g., GPT-4o vs GPT-4o mini).

Additionally, Google Colab allows direct interaction with the Gemini model within their workspace. You are welcome to use this feature as part of your assignment, but make sure to clearly annotate and document your interactions (i.e., prompts, responses, code changes).

Given the resource demands of language models, we will not require you to run the models locally. However, if you have the ability to do so, you are more than welcome to experiment with them! You may consider using libraries/tools such as HuggingFace Transformers, Ollama, or LM Studio. Please note the course staff will **not** be able to help troubleshoot locally run LMs, and insufficiently capable LMs may not yield useful outputs.

Prompting and Documentation

For all parts of this assignment, please clearly document:

- The LM being used.
- Your provided prompt(s).
- The LM output(s).

Due to the nature of LMs, there is no *singular* best solution. If you find a working prompt right away, you may instead ablate portions of the working prompt and report observations based on those changes. When working with the code, it is acceptable to manually identify issues first, and then work to develop prompts that help LMs specifically isolate and tackle the issues.

Assignment Overview

The Python code provided is intended to first generate a list of prime numbers, where their cube is less than or equal to some integer N. That is, first find the set S:

$$S = \{x \in \mathcal{Z} | x > 0 \wedge x^3 \leq N\}$$

Then, it should return up to the K largest values of S, as well as the actual number of primes returned:
A few helper functions are included for primality testing and generating prime numbers. The docstrings define the expected inputs/outputs of each function, but there are bugs in the code and the implementation is not optimized. Fully corrected, the outputs will match the expected test comments and the code will run significantly faster. If an LM suggests a better algorithm or approach, you are more than welcome to adopt it.

Note: the `assert` statements at in the *Testing and Validation* section of the Python notebook should not be modified. The code can be considered functionally correct once all assertions pass.

Assignment Parts

- Part 1: Given code, identify correction and optimization opportunities using an LM.
- Part 2: Develop a solution for the identified issues using an LM.
- Part 3: Iterate on prompts and explorations in Tasks 1 and 2 as needed, and develop a personal workflow for additional LMs.
- Part 4: Manipulate or subvert LMs via promptings to yield incorrect outputs.
- Part 5: Discuss strengths and weaknesses of LMs, what contributes to effective utilization and prompting, and how to formulate a clear use case. (**Do not** use LMs for this portion!)

Part 1 (10 points)

Diagnosing Code. Use an LM of your choice to identify *any* parts of the code to improve (e.g., incorrect functionality, performance issues, incorrect documentation, etc.). While something as simple as

"Is there anything wrong with this code? *[Code]*"

is enough for some models, please experiment with different prompts to see what works best!

- Document at least one issue identified by an LM, as well as the prompt you used to identify said issue(s). If an LM response includes multiple issues, modify the prompt to explicitly target individual issues.
- Was there an issue the LM did *not* find at first?

(Solution)

Example Writeup
(Please use this format for your writeup.)

Model: *HelpfulGPT-9001*
Prompt: Validate the functionality of `DownloadRAM`...
Response: Sure! Let me help you with that...
Found: Function `DownloadRAM` did not download RAM, due to a condition error in the *for* loop, caused by...

Model: *NotSoHelpfulGPT-8001*

Missed: There was a incorrect parameter type in the function `PrintMoney`...

Part 2 (20 points)

Fixing Code. Using the same LM that identified your reported issue(s) in Part 1, solve or improve the code based on the information you obtain. Try additional prompts or continuations (e.g., providing more details, asking for clarification or justification, etc.) and record your observations.

- Develop a solution to reported problem for Part 1. Document the prompt you used to obtain the solution. Make sure to explain how the solution works.
- Try at least 2 additional prompts or continuations (e.g., by adding or removing instructions, details, provided examples) to further improve or correct the proposed solution. Explain how the prompt was changed and how it impacted the the LM responses.

Part 3 (20 points)

Same But Different. Pick two other LMs apart from the one used already in Parts 1 and 2. Using each LM, identify a *different* issue (you might be able to create more targeted prompts based information from other LMs).

For each of the two additional LMs:

- Document the issue, as well as the prompt used to identify it (ala Part 1).
- Resolve the issue (ala Part 2).
- Were there specific issues each LM did *not* find?

Part 4 (15 points)

Poking Holes. Try to prompt LMs into providing an incorrect solution.

- Try to subvert at least 2 different LMs with up to 5-6 prompts/follow-ups each (fewer prompts are fine if you are successful). Document all prompts and responses.
- Were you able to successfully "confuse" or "trick" the LM?

Part 5 (35 points)

Reflection and Process Documentation. Now that you have had some hands-on experience with LMs in programming, we want you to consider when LMs can be useful, as well as when they may be ineffectual.

Please do not use LMs to come up with responses for this part of the assignment. The responses here should be your own thoughts, observations, and explanations.

Please answer the following questions about Parts 1-3, with supporting explanations/arguments:

- Which LM was the most helpful or easiest to prompt, and on what task? **(30 words max)**
- Were there any incorrect suggestions from an LM? **(30 words max)**
- Were there any tasks you felt you could do better or more efficiently with an LM? **(30 words max)**
- Were there any tasks you felt you could do more efficiently without an LM? **(30 words max)**
- What would you consider "essential" for effective prompting? Elaborate on prompting styles or specific phrasings that made LM responses more effective or robust. **(100 words max)**

Please answer the following questions about Part 4, with supporting explanations/arguments:

- Were there are any strategies you employed in subverting the LM? **(50 words max)**
- Which LM did you feel was easiest to manipulate, and why? **(30 words max)**

Feel free to note down any additional insights or comments here, as well as any questions you may have. Suggestions and feedback for this assignment are greatly appreciated!

Submitting the Assignment

Only **one submission per team** is needed. Your final submission should be a `.zip` archive named with a `CS2420_FA25_PSet3_` prefix followed by each team member's name separated by two underscores.

Example filename: `CS2420_FA25_PSet3_Name1__Name2__Name3__Name4.zip`

The archive should contain:

- PDF write-up
- Python notebook
- Any other relevant code (e.g., LM-generated snippets)
- Text files or PDFs containing the complete inputs to and outputs from of the generative AI tools used (e.g., exported ChatGPT logs).

Write-up

Written responses should be contained within a single PDF document. (LaTeX is highly recommended!) Responses and any figures should clearly indicate which part is being addressed. The write-up must contain the full names of all team members.