Assignment Specification: compare different prompting strategies and large language chat models for specified code-related tasks.

Parameters: The models I used were openai's 4o mini and a preview of and Gemini's 2.5 flash, and the techniques I used were zero-shot and chain-of-thought. The metrics I used for quantitative evaluation were BLEU, ROUGE, and CodeBert. Prompt design: I used the given prompts as a base to work off of, supplementing with some data I learned from reading the given code skeletons (but I incorporated no data into the prompts from the skeletons that I wanted the model to figure out). I designed the prompts in a somewhat systematic way. They all begin with the essential verb asking what the model should do. They all refer to the programming language that they should use to reply. They all are well-formed sentences. The only action I took to create the chain-of-thought prompt from the base (zero-shot) prompt was to insert the sentence "Explain your reasoning in a logical progression of steps." In making the difference between my prompts this simple, I aim to answer the question of whether chain-of-thought is an improvement over zero-shot.

Workflow: I had ChatGPT generate (using prompt chaining!) a json file where I could paste all my responses, as well as Python scripts that I could execute to request and import the responses from API, compute the relevant metrics and format all the data into a human-readable xlsx file. I entered the prompts in the models as described in the control section, and ran the Python scripts.

Outputs: I have the language outputs (the response of the models, as well as just the code from the response) in responses_comparison_two_updated.xlsx. I have the comparison metrics in model_comparison_metrics. I mistakenly doubled the comparison by adding columns for both the openai and gemini responses - the point of the metrics, since we don't have access to the ground truth, is to compare them to each other, so of course the columns are the same.

Individual (local) analysis: here we discuss the differences in the four outputs per each task. The tasks are ordered by their order in the project specification, and the line given for each of the 20 tasks is the base prompt. The zero-shot prompt is base prompt + the given code in the project spec. The chain of thought prompt is base prompt + "Explain your reasoning in a logical progression of steps." + the given code. The code was extracted from outputs and compared with the metrics: OpenAI/Gemini comparisons for both prompts, and comparisons across prompts within models (e.g. OpenAI Zero Shot/OpenAI Chain of Thought).

1. Summarize in a few sentences the functionality of the following Java method.
    a. Metrics:
    b. Qualitative:
2. Identify and fix the off-by-one error in the following Python function.
    a. Metrics:
    b. Qualitative:
3. Classify the type of bug present in the following function.
    a. Metrics:

    b. Qualitative:
4. Complete with regex the following function, which will validate email addresses.
    a. Metrics:
    b. Qualitative:
5. Create a ‘/greet/‘ endpoint API in Python that returns a JSON greeting by completing the following function.
    a. Metrics:
    b. Qualitative:
6. Write the schema for an app that allows users to review books that would naturally follow the included comments below.
    a. Metrics:
    b. Qualitative:
7. Identify any null deference risk by filling out the following Java skeleton.
    a. Metrics:
    b. Qualitative:
8. Improve the parser based on the following Python skeleton in order to support quoted fields.
    a. Metrics:
    b. Qualitative:
9. Convert the Kotlin data class below to a REST API using Ktor.
    a. Metrics:
    b. Qualitative:
10. Summarize in a few sentences the functionality of the following Python method.
    a. Metrics:
    b. Qualitative:
11. Write a prompt that an AI model like yourself could use to generate the code below.
    a. Metrics:
    b. Qualitative:
12. Fix the bug in the following Python code for the factorial function that happens when the input is 0.
    a. Metrics:
    b. Qualitative:
13. Implement in C using the following skeleton a method to delete nodes in a linked list.
    a. Metrics:
    b. Qualitative:
14. Complete the following empty Python method that implements Fibonacci numbers recursively.
    a. Metrics:
    b. Qualitative:
15. Complete the empty Python class constructor given below.
    a. Metrics:
    b. Qualitative:
16. Complete the following partial Java implementation of binary search.
    a. Metrics:

    b. Qualitative:
17. Resolve inconsistency between the name and the logic of the following C++ function.
    a. Metrics:
    b. Qualitative:
18. First identify the bug in the following JavaScript code. Then fix it.
    a. Metrics:
    b. Qualitative:
19. Decompose the high-level content/summary of the following C++ code into logical steps.
    a. Metrics:
    b. Qualitative:
20. Complete the following skeleton for a Python method based on the intent of the function.
    a. Metrics:
    b. Qualitative:

Global analysis: here we discuss variances between the models and the prompting strategies seen across tasks.