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 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 and CodeBert (embedding similarity). ROUGE was also computed, but on the code only - ROUGE is best applied to text summarization, not code, so we ignore it.
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_four.xlsx. I have the comparison metrics in model_comparison_four.xlsx. The metrics were evaluated as follows: for each task and prompt (that results in a code output - some don't), compare the OpenAi and Gemini responses to each other to get BLEU and CodeBert. We also compare the OpenAi zero shot vs chain of thought responses to yield metrics, for both prompts, and likewise with Gemini.

Global analysis: here we discuss variances between the models and the prompting strategies seen across tasks. Only traits not discussed in this section are discussed in the section on individual methods.

BLEU zero shot tends to outperform BLEU chain of thought, so the zero shot responses from OpenAI and Gemini tend to be more similar than chain of thought responses from the two models. There are exceptions, and the metrics are often low, not passing .5 more than a few times. This suggests that a chain of thought prompt is more likely to depend on the differences in how the models "think", and that those differences affect the functional output.

In comparing zero shot to chain of thought, BLEU score tends between .2 and .7 with minor exceptions, for both OpenAI and Gemini. OpenAI scores on average somewhat higher. This indicates that OpenAI is slightly more consistent among these prompting strategies and so its internal logic is slightly stronger. The logic has a slightly lower temperature.

CodeBert scores for embedding were high across the board, often in the .9+ range. The zero-shot comparison between models tended to fall between .96 and .99. The chain of thought comparison had a few lower outliers but tended between .94 and .99. This indicates that zero shot prompts produce results that are slightly more similar to each other across models than chain of thought prompts.

CodeBert scores for OpenAI zero shot vs chain of thought tended above .98, but with a good number of outliers. For Gemini, it was a more even distribution around the .9 range. This indicates that OpenAI was more consistent with its outputs across prompting strategies than Gemini, and that Gemini again was somewhat variable.

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.
   OpenAI referenced the name of the method, while Gemini referenced the method by "Java method". They both gave essentially correct answers every time. The OpenAI Chain-of-thought case ended up discussing the efficiency of the data structure a lot. The Gemini explanation was more technical.
2. Identify and fix the off-by-one error in the following Python function.
   Every response indicated to some extent that the code was already correct. Both of the OpenAI models addressed the possibility that $n<0$ is an input and changed the code to account for this.Both Gemini responses outright said that the prompter was wrong and there was no error. The zero shot prompt mentioned later on that this could be an error, counterintuitively, Gemini did not mention a $n<0$ case anywhere in the chain of thought. This indicates that Gemini likes to "stick to its guns" - if the model 'believes' something, it will not try to consider other cases.
3. Classify the type of bug present in the following function.
   All four cases behaved pretty similarly. The zero shot openai case used sentences, but all other cases defined logic steps in its argument and discussed those steps. The logic was very similar.
4. Complete with regex the following function, which will validate email addresses.
   Gemini was much more verbose, and took many steps in its explanations for both prompts. Gemini zero shot and OpenAI chain of thought explained meaning of various symbols in an integrated manner with its logic, while the other two cases did this separately.

5. Create a ‘/greet/‘ endpoint API in Python that returns a JSON greeting by completing the following function.

    All models defer an explanation of the given code to the end. Gemini is more verbose. OpenAI zero shot is the only case that does not explain logic layers to quite the extent the other cases do.
6. Write the schema for an app that allows users to review books that would naturally follow the included comments below.

    The responses from OpenAI are quite similar to each other as are the ones for Gemini. OpenAI in both cases explains the basic elements of the response, but Gemini explains every single parameter of each of the elements.
7. Identify any null deference risk by filling out the following Java skeleton.

    Gemini used more steps to explain, and chain of thought Gemini used the most. Gemini explained its reasoning more thoroughly and technically than OpenAI. The code generated was pretty similar.
8. Improve the parser based on the following Python skeleton in order to support quoted fields.

    The OpenAI responses were completely different. In zero shot, OpenAI wrote a short module and explained it; in chain of thought, the model explained everything one should do to improve the parser and implement each of those steps into a long method. Gemini followed this pattern too. Its outputs were longer and more detailed, and it explained quite a lot via code comments.
9. Convert the Kotlin data class below to a REST API using Ktor.

    OpenAI zero shot gave a few steps and an example; chain of thought did the same but elaborated on steps along the way. The Gemini responses read kind of like a Jupyter notebook, and Gemini’s explanation was an explanation of every argument/variable of the script it wrote. Gemini’s chain of thought response was incredibly long - 11 steps with about 50 lines each.
10. Summarize in a few sentences the functionality of the following Python method.

    All four responses looked pretty similar, being only a few sentences long. The chain of thought responses were slightly longer/more detailed. Only the OpenAI Chain of Thought response gave numbered steps.
11. Write a prompt that an AI model like yourself could use to generate the code below.

    All responses were reasonably similar. Only OpenAI zero shot did not provide a detailed explanation. Gemini zero shot actually gave a few different prompts that reveal different info.
12. Fix the bug in the following Python code for the factorial function that happens when the input is 0.
13. Implement in C using the following skeleton a method to delete nodes in a linked list.

    Gemini gave sample uses and OpenAI did not. OpenAI explained its high level reasoning (for both prompts) and Gemini went very in depth on the level of the code, not necessarily the broader logic.
14. Complete the following empty Python method that implements Fibonacci numbers recursively.

Gemini's analysis was much more on the level of what should happen next in the function it is writing. It explains the function it wrote as it writes it in both prompts. OpenAI gives a high level summary in both cases, which is longer in the chain of thought case. Gemini explained the n<0 input problem and OpenAI did not.

15. Complete the empty Python class constructor given below.

This example was the ideal demonstration in how these models respond to prompts differently. The OpenAI zero shot case just gave a solution and a one line explanation. The OpenAI chain of thought case gave high level reasoning to accompany its solution. The Gemini zero shot case gave a solution with some reasoning, but the reasoning was essentially an explanation of the specific lines of code. Gemini chain of thought gave high level reasoning.

16. Complete the following partial Java implementation of binary search.

    The responses were all remarkably similar - the lowest CodeBert embedding score for these responses was >.98. OpenAI chain of thought explained steps and then gave code, and all other cases did the opposite.
17. Resolve inconsistency between the name and the logic of the following C++ function.

    All four responses were similar and directly identified the two possible fixes. They all provided some number of steps (OpenAI zero shot gave 2, Gemini chain gave 7) for the logic used to come to this conclusion.
18. First identify the bug in the following JavaScript code. Then fix it.

    The responses all took some variant of explaining the error and giving a better solution. All but OpenAI zero shot explained why it was better using reasoning. The reasoning the models explained ended up just being explanations of the tools they used to come to the code, so it wasn't a true explanation of reasoning.
19. Decompose the high-level content/summary of the following C++ code into logical steps.

    The Gemini zero shot case didn't explain steps to the extent the other cases did. All the other cases interpreted "explain your reasoning" as explaining why each step they took was correct with respect to the known rationale of the factorial function. Gemini chain of thought gave both an explanation of what was happening and the reasoning behind the step (i.e. the known rationale of the factorial).
20. Complete the following skeleton for a Python method based on the intent of the function.

    All models essentially decided to work through the steps a human would need to take to complete this and explain those steps. Gemini was more verbose and had more steps. The Chain of Thought Gemini response actually used "explain your steps" as a step in its overall logic, explaining the logical progression of the function as it wrote it in that step.

Conclusion: Here are some key takeaways from this analysis:

- OpenAI has a stronger internal logic than Gemini and a greater ability to spot fringe cases. OpenAI likes to give a big picture, and only elaborates further to a deep level when prompted.
- Gemini is more verbose, and better for detailed explanations, but only to the extent that it is able to give correct outputs. Gemini likes to explain every token. Gemini also tends to

give its reasoning without chain of thought prompts, but chain of thought increases the specificity.

- Chain of thought substantially increases the output by OpenAI, which otherwise reverts to concise explanations. Gemini is made even more verbose and precise when chain of thought is applied.