GenAI for Software Development: Assignment 3

Quang Hoang
qhoang@wm.edu

1 Introduction

Prompt Engineering for In-Context Learning investigates the impact of different prompt designs on the performance of Large Language Models (LLMs) across a variety of software engineering tasks. In this assignment, five prompting strategies—zero-shot, few-shot, chain-of-thought, prompt-chaining, and self-consistency—were applied to 22 tasks, including code summarization, bug fixing, API generation, and code translation. Experiments compared four models—**gpt-4.1**, **Codestral-2501**, **gpt-4.1-mini**, and **gpt-4.1-nano**—to demonstrate how the strategic use of prompt examples and structured reasoning influences the quality and clarity of generated code. Self-consistency prompting employed three repetitions, with a temperature setting of 0.7 and a maximum token limit of 1024 tokens across all evaluations.

In this assignment, I explore the power of in-context learning by designing effective prompts to perform various software engineering tasks. I craft and compare different prompting strategies for a diverse set of problems using large language models (LLMs). The LLMs to be used are:

- OpenAI GPT-4.1 (gpt-4.1)
- Codestral 25.01 (Codestral-2501)
- OpenAI GPT-4.1-mini (gpt-4.1-mini)
- OpenAI GPT-4.1-nano (gpt-4.1-nano)

The source code for my project can be found at
https://github.com/theantigone/Prompt-Engineering-for-In-Context-Learning

2 Implementation

Model Token Generation: To test different prompting strategies, I use **Azure**'s **OpenAI Service** at GitHub Marketplace to access the models that I use for my assignment. Then, I use **OpenAI**'s pip package to run the models using my GitHub personal access token. To organise information, I use **pandas** DataFrames, which were used to store prompts, code snippets, and model responses.

Model Evaluation: To evaluate the models' outputs, I use various evaluation metrics, such as **BLEU**, **ROUGE**, and **METEOR** for natural language, and **CodeBLEU** and exact matches for code. I also implement **cosine similarity** to test **embedding-based similarity**. Many pip packages are necessary for the evaluation phase.

3 Observations

Varying Strengths: Different models exhibited strengths in different areas. Some were more concise, while others were more thorough with details and examples.

Prompt Influence: The effectiveness of models was influenced by the prompting strategy (zero-shot, few-shot, chain-of-thought, prompt-chaining, self-consistency). Chain-of-thought generally led to more detailed explanations, while few-shot emphasized conciseness and template adherence.

Detail vs. Conciseness: A recurring theme was the trade-off between detailed explanations and concise outputs. Some models tended towards verbosity, ensuring clarity, while others prioritized brevity, which could sometimes lead to a loss of context.

Task Specificity: Model performance also depended on the specific task, such as code summarization, bug fixing, schema design, or API generation.