Assignment 1

- Course: COMP7607: Natural Language Processing - The University of Hong Kong
- Semster: Fall 2025
- Due: October 26

Recent studies reveal that large language models (LLMs), such as Llama (Dubey et al., 2024) and GPT-4 (OpenAI, 2023), excel in step-by-step reasoning for addressing complex tasks driven by natural language instructions. In this assignment, you will explore the capabilities of LLM in two reasoning domains: mathematics and coding. You can choose one task—either mathematics or coding—based on your interests. Submit: You should submit your assignment to the COMP7607 Moodle page. You will need to submit a PDF file UniversityNumber.pdf of your report (with detailed experimental details) and a zip file UniversityNumber.zip, which includes:

- .py files.
- zeroshot.baseline.jsonl
- fewshot.baseline.jsonl (for math task)
- [method_name_1].jsonl
- [method_name_2].jsonl
- [method_combine].jsonl or [method_improve].jsonl

Please note that the UniversityNumber is the number printed on your student card

1 Introduction

Prompt Engineering refers to methods for how to instruct LLMs for desired outcomes without updating model weights. The core task of assignment 1 is to design methods for prompting LLMs to improve the accuracy of LLMs on math problem-solving or code generation.

Generally, we have two prompting paradigms:

1. **Zero-shot prompting** is to simply feed the task text to the model and ask for results. For example, we can send the following message to:

```
messages = [
    {
        "role": "system",
        "content": "[System Message]"
    },
    {
        "role": "user",
        "content": "Elsa has 5 apples. Anna has 2 more apples than Elsa. How many apples do they have together?"
    }
]
```

`{"role": "system", "content": "[System Message]}"}` can be optional. [System Message] is a feature-specific contextual description given to a generative AI model, such as

`Your task is to solve a series of math word problems by providing the final answer. Use the format #### [value] to highlight your answer.`

2. Few-shot prompting presents a set of high-quality demonstrations, each consisting of both input and desired output, on the target task. As the model first sees good examples, it can better understand human intention and criteria for what kinds of answers are wanted. For example, we can send the following 1-shot prompting message to LLM:

```
messages = [
    {
        "role":"system",
        "content": [System Message]
    },
    {
        "role":"user",
        "content":"Elsa has 5 apples. Anna has 2 more apples than Elsa. How many apples do they have together?"
    },
    {
        "role": "assistant",
        "content": "Anna has 2 more apples than Elsa. So Anna has 2 + 5 = 7 apples. Elsa and Anna have 5 + 7 = 12 apples together. ####
    },
    {
        "role": "user",
        "content": "If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?"
    },
]
```

The choice of prompting format, interaction method, and decoding parameters will all significantly influence task performance. Now enjoy exploring the magic of prompting with LLM!

2 Task

For your selected task (math or code), begin by implementing the two simplest baselines: zero-shot and few-shot prompting, using only the problem statement with demonstrations (optional). Please refer to the example format in Section 1. Then, choose two advanced methods to enhance performance
beyond baselines. We present advanced methods that you may choose to implement in Section 2.1 and Section 2.2, respectively. You are also encouraged to propose your own designs!

Here, we provide two surveys that offer a comprehensive overview of the development of prompting engineering and the self-correction of LLMs:

- A survey for prompt engineering for autoregressive language models Survey Link.
- A collection of research papers for LLMs self-correcting Survey Link. Note that we only focus on the "Generation-Time Correction" and "Post-hoc Correction".

2.1 Mathematical Reasoning

Data We use the grade school math dataset, GSM8K, which is one of the most popular datasets for evaluating the mathematical reasoning performance of LLMs. Note that the performance of the LLM is measured using "GSM8K/test.jsonl". You can utilize "GSM8K/train.jsonl" for other objectives, such as retrieving the most similar examples for the questions in the test file.

Metric For each method, report the overall accuracy on the 1,319 questions in the test file. Use the answer extraction function provided in "GSM8K/evaluation.py" for calculating accuracy. Additionally, report the inference cost using two metrics: wall-clock time and the average number of generated tokens per math question.

Task 1: Implement Baseline We consider two baselines:

- Zero-shot prompt the model using only the problem statement.
- Few-shot prompts the model with a few demonstrations, each containing manually written (or model-generated) high-quality reasoning chains.

We provide the prompting template in "GSM8K/baseline.py".

Task 2: Implement two Advanced Prompting Methods You may explore various advanced strategies for improvement, such as retrieving similar demonstrations, decomposing responses, employing response ensemble, engaging in self-improvement, and more. Below, we present several advanced methods, although not exhaustive, that you may choose to implement, considering the characteristics of mathematical reasoning tasks. You are also encouraged to propose your own designs!

- Self-Polish: Enhance Reasoning in Large Language Models via Problem Refinement paper
- Tree Prompting: Efficient Task Adaptation without Fine-Tuning paper
- SELF-REFINE: Iterative Refinement with Self-Feedback paper
- PHP: Progressive-Hint Prompting Improves Reasoning in Large Language Models paper
- CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing paper
- Teaching Algorithmic Reasoning via In-context Learning paper
- Contrastive Decoding Improves Reasoning in Large Language Models paper
- Self-Evaluation Guided Beam Search for Reasoning paper
- Skills-in-Context Prompting: Unlocking Compositionality in Large Language Models paper
- Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks paper
- Large Language Models are Better Reasoners with Self-Verification paper

Task 3: Combine the Two Methods or Enhance One Method Can you combine the two advanced methods or enhance one method based on your analysis to achieve greater gains? Are complementary or enhanced effects achievable? If not, please explain why.

2.2 Code Generation

A particularly intriguing application of LLMs is code generation, a task that involves producing source code from natural language descriptions. This area has garnered substantial interest from both academia and industry, as evidenced by the development of tools like GitHub Copilot.

Data We use HumanEval leaderboard, which has been established as a de facto standard for evaluating the coding proficiency of LLMs. Note that the performance of the LLM is measured using "HumanEval/HumanEval.jsonl".

Metric For each method, report the overall accuracy on the 164 questions in the "HumanEval.jsonl" file. We provided "example_problem.jsonl" and "example_solutions.jsonl" under HumanEval to illustrate the format and assist with debugging. You can evaluate the accuracy of the generated code by running:

```
python evaluate_functional_correctness.py -sample_file example_samples.jsonl -problem_file example_problem.jsonl
```

Also, report the inference cost using two metrics: wall-clock time and the average number of generated tokens per code problem.

Task 1: Implement Baseline We use zero-shot prompting as the baseline. The prompting template is provided in "HumanEval/baseline.py".

Task 2: Implement two Advanced Prompting Methods You may explore various advanced strategies for improvement. Below, we present several advanced methods that you may choose to implement. You are also encouraged to propose your own designs!

- Teaching Large Language Models to Self-Debug paper
- SelfEvolve: A Code Evolution Framework via Large Language Models paper
- Self-Edit: Fault-Aware Code Editor for Code Generation paper
- Lever: Learning to verify language-to-code generation with execution paper
- Is Self-Repair a Silver Bullet for Code Generation? paper
- Codet: Code generation with generated tests paper
- SELF-REFINE: Iterative Refinement with Self-Feedback paper
- Language agent tree search unifies reasoning acting and planning in language models paper

Task 3: Combine the Two Methods or Enhance One Method
Can you combine the two advanced methods or enhance one method based on your analysis to achieve greater gains? Are complementary or enhanced effects achievable? If not, please explain why.

Task 4: Report the Inference Cost

In this assignment, you need to report the token usage for each task. Specifically, you should record:

- Input tokens: The number of tokens in your prompts (including system messages, user queries, and context)

- Output tokens: The number of tokens generated by the model in its responses
- Total tokens: The sum of input and output tokens

Please document these metrics for each task in your report. Most LLM API providers return token counts in their response objects, or you can find this information in your API usage dashboard. This will help you understand the computational cost and efficiency of different prompting strategies.

3 Model and API

In this assignment, you can use any LLM model to complete the tasks. You can get free token credits (Note: Your free tokens are limited. If you encounter API calling errors, check whether you've exceeded your quota.) on the Alibaba Cloud Bailian platform: https://www.alibabacloud.com/en?_p_lc=1/ (You can find the guide of Bailian platform here, (CN) and here is an example API tutorial to help you get started). Alternatively, you're welcome to purchase your own tokens from providers like OpenAI, DeepSeek, Grok, Gemini, or Claude to complete the assignment. You can find API usage guides for these LLM providers on their official websites.

4 Report

You will write a report including the following parts:

- The description of your implemented methods, including the architecture, the hyperparameters, etc.
- The discussion and table of your results, including the hyperparameters you have tested, the performance of the methods on the test set, the analysis of the results, the advantages compared to the baselines, etc.

5 Note

There are some key points you should pay attention to:

• Your assignment will not be evaluated based solely on the accuracy of the task. Instead, we primarily assess your work through your experiments and analysis.
• You can adjust the decoding hyperparameters to get a better performance on the test set. The hyperparameters include the temperature, top-p, top-k, etc.