Assignment 1 Report

Chenxi Peng

October 22 2025

1 Task 1: Evaluation Harness

In part 1, I referred to `run_model.py` to parse command-line arguments. Then I define some functions. They can achieve the following purposes respectively. 1. extracts `#### NUMBER` from the answer and convert it into float; 2. creates the prompt with the required output format; 3. invokes a reasoning model. Finally, I downloaded the GSM8K dataset from Hugging Face and examined each example from the dataset to obtain the accuracy, time consumption, and output word count of the reasoning model on this mathematical reasoning benchmark. Then calculate the average and variance of the time taken and the tokens. My prompting is "`Question: {question}\n\n Answer: Put your final numerical answer in the format: #### NUMBER`", so the expected output format is "`#### NUMBER`".

Running the `part1.py` with three different LLMs, GPT-OSS-20B, GPT-OSS-120B, and Gemma-3-27B-IT, from the Table 1, we can see that the first two models performed well on the math reasoning benchmark, but Gemma performed poorly. So I used gpt-oss-20b to perform Task 2 and Task 3. GPT-OSS-120B is a larger, more powerful model designed for heavy-duty tasks, while GPT-OSS-20B is a smaller, more efficient model optimized for local and on-device use. Since the performance of these two LLMs is similar, and GPT-OSS-20B is a smaller, more efficient model optimized for local and on-device use, I use GPT-OSS-20B to perform Task 2 and Task 3.

Table 1: The comparison between three LLMs on the math reasoning benchmark

Metric	**openai/gpt-oss-20b**	**openai/gpt-oss-120b**	**google/gemma-3-27b-it**
Overall Accuracy (%)	100	100	50
Latency (seconds)			
Average	2.40	2.38	3.87
Std. Dev	0.84	0.69	1.25
Output Tokens (tokens)			
Average	2.00	2.00	106.65
Std. Dev	0.00	0.00	60.37

Note: The number of instances is 20, taken from the `main_train` branch of the GSM8K dataset. Output tokens are calculated by counting the words in each generated response.

2 Task 2: Implement Chain of Thought Prompting

I used 10 instances from the Hugging Face by querying "`SELECT * FROM main_train LIMIT 10`" to construct a sample list used in the chain of thought. From the Table 2, we may think that with the increasing number of samples, the overall accuracy will show a log-like graph, meaning that the marginal improvement to the total accuracy will decrease when the number of samples in the chain of thought (COT) increases. Actually, I think it is consistent with my expectation - the structure of COT matters more than the number of samples in COT. With the increasing number of samples in COT, the output tokens decrease, and the standard deviation drops dramatically, meaning that the performance becomes stable.

Table 2: Evaluation of GPT-OSS-20B with the COT: variation in the number of samples in COT

Metric	**1 Sample**	**5 Samples**	**10 Samples**
Overall Accuracy (%)	95	100	100
Latency (seconds)			
Average	2.26	2.58	2.18
Std. Dev	0.75	0.89	0.45
Output Tokens (tokens)			
Average	13.10	13.15	2.00
Std. Dev	20.93	26.51	0.00

Note: The number of instances is 20, taken from the `main_train` branch of the GSM8K dataset. Output tokens are calculated by counting the words in each generated response.

To figure out the effect of the structure of the samples on the performance of the model, I changed the structure of the samples into extremely simple ones and used 5 samples in the COT. For example, the original one is "`Natalia sold 48/2 = «48/2=24»24 clips in May.Natalia sold 48+24 = «48+24=72»72 clips altogether in April and May.#### 72`", while the simplest is "`In April Natalia sold 48 clips.In May Natalia sold 24.  72 altogether in April and May.  #### 72`". From the Table 3, we can see that COT with less detailed examples makes the latency and tokens in the output improve. I think it is consistent with the expectation, because the agent will refer to the examples to answer the question.

Table 3: Evaluation of GPT-OSS-20B with the COT (5 samples): variation in the structure of the samples

Metric	**Original**	**Minimalist**
Overall Accuracy (%)	100	100
Latency (seconds)		
Average	2.58	2.16
Std. Dev	0.89	0.65
Output Tokens (tokens)		
Average	13.15	5.55
Std. Dev	26.51	12.15

Note: The number of instances is 20, taken from the `main_train` branch of the GSM8K dataset. Output tokens are calculated by counting the words in each generated response.

3 Task 3: Implement a ReAct Loop

To build a ReAct agent, I use `LangChain`. Table 4 shows the comparison of the agent performance with three frameworks. According to common sense, the ReAct agent should perform better, or at least not worse than the previous two frameworks. From the result, we can see that ReAct agent will generate more output tokens than the first two agents. The reason may be that in ReAct loop, the agent will implement two processes: think and action, causing the tokens to become huge. In this math reasoning case, I think using the tool isn't needed, because the math questions here are relatively simple. There is no need to consume more tokens.

Table 4: Evaluation of GPT-OSS-20B: variation in the framework of the agent

Metric	**Original**	**COT (10 slots)**	**ReAct loop**
Overall Accuracy (%)	100	100	100
Latency (seconds)			
Average	2.40	2.18	2.92
Std. Dev	0.84	0.45	2.03
Output Tokens (tokens)			
Average	2.00	2.00	220.40
Std. Dev	0.00	0.00	319.76

Note: The number of instances is 20, taken from the `main_train` branch of the GSM8K dataset. Output tokens in the first two frameworks are calculated by counting the words in each generated response, while in ReAct loop the number of output tokens is extracted from the metadata in the result of the function.