# **Report**

## **1. Introduction**

**1.1 Task Overview**

This task is going to dive LLM up and utilize the most likely ability of LLM, by find the correct answer in the multi-choice question without any fine-tuning or retraining process.

**1.2 Background and Related Work**

There has two kinds of QA task, one is deduce the answer from answer, another is choose the word that is in the question, and extract this word as answer. Before Vanilla Transformers, people used RNN to do text classification task to implement multi-choice question, which means that practitioner needs to train lots of models for different domain. After the advent of Vanilla Transformers and BERT, the paradigm of “fine-tuning” was coming up, which only asks people prepare a small piece of dataset from specific domain and fine-tune the pre-train model on these data. It saves lots of time to train a model from scratch and the performance of “fine-tunning BERT” is SOTA. Since the number of model parameters is becoming bigger and bigger, more and more people couldn’t afford money to train or fine-tune an LLM. So this phenomenon boosts the “prompt tuning” paradigm. Only give the LLM the prompt to inference the result.

## 2. Methods

## **2.1 baseline**

## the whole procedure could be considered as the following step:

1. feed the whole training dataset via **bge model** get a 2D tensor.
2. feed every validation dataset via bge model and calculate the inner product with the tensor from step 1(the more inner product it is, the more closed/similar these two tensors have) as the score, sort the score and choose the most likely N sample from training dataset.
3. build the prompt from 2 styles of prompt and use LLM to inference the result.

**2.2 improvement**

**What I improve:**

1. Change the function of generating prompt. It only receives a parameter of max\_len and tries to fill the prompt to max\_len-100 as closed as it can, not just specify the number of question-answer pairs. Since the answer won’t be too long, so no useful information will be truncated during encoding or decoding.

2. Use reverse parameter to format the prompt. If reverse is true, then the max similar they are, question-answer pair appear first, and vice versa.

3. Find the gap regarding performance of phi1.5 and phi2.

4. Compare the performance of embedding model between bge-little, bge large, bge-m3, llm-encoder.

4. Linear interpolation to implement length extrapolation. (It requires much more memory, like 32GB to run it, GIVE UP)

5. Hyperparameters tuning

6. Fix some bugs

**3. Experiment**

**3.1 Origin model:**

the performance of origin model is as follows: (eval\_few\_shot.py)

|  |  |
| --- | --- |
| **easy + max\_len 1024 + prompt 2 + N 8 + reversed False** | **0.80175** |
| **easy + max\_len 1024 + prompt 1 + N 8 + reversed False** | **0.62982** |
| **easy + max\_len 1024 + prompt 2 + N 8 + reversed True** | **0.78771** |
| **challenge + max\_len 1024 + prompt 2 + N 8 + reversed False** | **0.5217** |
| **challenge + max\_len 1024 + prompt 1 +N 8 +reversed False** | **0.4414** |
| **challenge + max\_len 512 + prompt 1 + N 8 + reversed False** | **0.2140** |

**After doing experiments, I found that the following conclusion:**

* **the more related training examples in the prompt, the more performance it gets.**
* **prompt 2 has better performance than prompt 1 has.**
* **reversed True affects LLM inferencing process, LLM might be likely to process the most similar text first.**

Firstly, find the reason why the baseline doesn’t work well, talk about what baseline might miss and then take action to improve.

* the hyperparameters N and max\_len might collide with each other, which means that if N is bigger and max\_len is small, tokenizer will truncate those last characters. Based on the prompt you defined, the model won’t know what you want it to inference.
* Does Phi1.5 really have the capability to inference?
  + Understand that question might help us figure out why prompt 2 has better performance in Phi1.5 than prompt 1 has. Since prompt 2 doesn’t have candidate answer information but it has better performance in phi1.5, it’s normal that prompt 1 should be better than prompt 2.
  + Just execute **tokenizer.batch\_decode(output)** to print what Phi1.5 inference and find that the output is full of “computer language code” and Phi1.5 just wants to write code under any other prompts, which is really weird, because the prompt doesn’t contain any code.
  + So I can deduce that phi1.5 might don’t have enough ability to handle this task. I changed phi1.5 to phi2 and check the output after feeding the prompt, the output is quite reasonable, although it contains some repeated text.
* Based on the specified hyper-parameters set, do I choose the most similar come first or the less similar come first in the prompt?
  + After experimenting, it’s better to show the most related QA pair first, which might teach LLM to learn at the right direction.

**3.2 Improvement**

The improvement I did has been shown in the part 2.2. I will illustrate the performance of experiment directly.

Still use Phi1.5 and only change the way that building prompts. After filling as many similar training data pairs as it can to the prompt, the performance is better in the easy dataset and worse in the challenge dataset, compared with the baseline.

|  |  |
| --- | --- |
| easy + max\_len 1024 + modified prompt 2 + reversed + phi1.5 | 0.7912 |
| easy + max\_len 1024 + modified prompt 1 + reversed + phi1.5 | 0.6070 |
| easy + max\_len 2048 + modified prompt 2 + reversed + phi1.5 | 0.8140 |
| easy + max\_len 1024 + modified prompt 2 + not reversed + phi1.5 | 0.7894 |
| challenge + max\_len 2048 + prompt 2 + revered + phi1.5 | 0.4983 |

After executing tokenizer.batch\_decode(output), the phi1.5 inferences some code in the outputs, which is ridiculous, because there doesn’t have any code in the prompt. It’s time to change phi1.5 to phi2. Firstly, test the performance on the baseline, just change the model.(eval\_fewshot.py)

|  |  |
| --- | --- |
| easy + max\_len 1024 + prompt 2 + N 8 + not reversed + phi2 | 0.8438 |
| easy + max\_len 1024 + prompt 2 + N 8 + not reversed + phi1.5 | 0.80175 |
| challenge + max\_len 1024 + prompt 2 + N 8 + not reversed + phi2 | 0.5752 |
| challenge + max\_len 1024 + prompt 2 + N 8 + not reversed + phi1.5 | 0.5217 |

The performance has been increased significantly after using phi2, compared with phi1.5. So the following experiments will be under phi2 + changing the way of build prompts. (improvement\_few\_shot.py)

|  |  |
| --- | --- |
| easy + bge-small-v1.5 + modified prompt 2 + max\_len 1024 + phi2 | 0.8473 |
| easy + bge-small-v1.5 + modified prompt 2 + max\_len 2048 + phi2 | OOM (16GB) |
| challenge + bge-small-v1.5 + modified prompt 2 + max\_len 1024 + phi2 | 0.5986 |
| challenge + bge-small-v1.5 + modified prompt 1 + max\_len 1024 + phi2 | 0.7625 |
| easy + bge-small-v1.5 + modified prompt 1 + max\_len 1024 + phi2 | 0.8842 |

Since prompt 1 has more information about candidate answer, it should be better to inference the answer. However, under phi1.5 model, the performance of prompt 1 is much lower which means that Phi1.5 model doesn’t have enough ability to tackle this problem. After changing the model to Phi2, the performance of prompt 1 is much better than the performance of prompt 2.