CSE 587 Final Project:

KYEONG-HOON LIM

Github Link: https://github.com/kyeongHoony/CSE587

1. Problem Definition and Dataset Construction

Problem Definition: In academic papers, there are many sections such as abstract, introduction, background, motivation, methodology, etc. Among these sections, the abstract section plays an important and unique role. This is because, as the goal of abstract section is to introduce the papers to readers and make them interested in the papers, abstract section is a brief summary of papers and contains important contents of paper in a compressed way. Therefore, correctly understanding the abstract section is important to get a brief picture of a paper before we start to read the full paper. The importance of accurately appreciating the abstract section escalates during the literature search, where researchers have to analyze a large number of papers related to their topic.

For this reason, in this project, I will make an LLM model which is fine-tuned for the task of properly understanding the abstract of papers. For fine tuning, I used academic papers in computer architecture. I chose one of the prestigious conferences in this field, *The International Symposium on Computer Architecture (ISCA)* [1], as a source of academic papers. Naively checking whether the given LLM model understands the given input correctly or not is a difficult problem. Therefore, to establish a quantitative score system, I adopted question and answer based evaluation model. The Question and answer pairs are made by the LLM model based on the given abstract. For training dataset, the question and answer pairs were generated by Gemma LLM model which has 2B parameters [2]. For test dataset, the QA pairs were created using Mistral LLM model which has 7B parameters [3]. The Figure1 shows brief summary of my project.

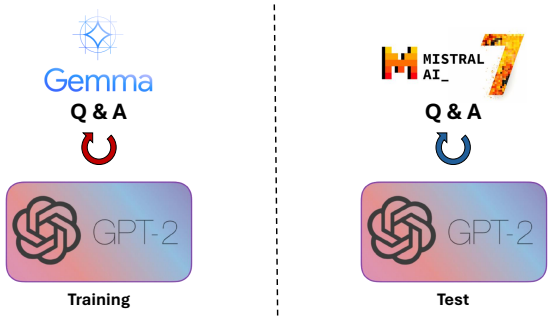

Fig. 1. Overview of fine-tuning and test process.

Gathering Data: To gather the raw data, I retrieved HTML files using *get* command. The Python code for this web crawling is uploaded in the GatherData directory (crawling.py). In this code, I first obtain URLs from DBLP site and I access sites using these URLs. However, some HTML files didn't clearly provide the abstract contents. To deal with these cases, I used the Selenium library. The Python program using this library is uploaded in the GatherData directory (crawling2.py). The format of raw data is shown in Figure2.

The total number of papers is 1,144. For training, I used papers published between 2005 and 2021, and for testing, I used papers published between 2022 and 2024. The trend in the number of papers published each year is shown in Figure 3.

Making QA pairs using LLM model: To pre-process the raw data, I used two different LLM models: Gemma [2] and Mistral [3]. To have these LLM models to generate question and

```
{
  "title": "GhOST: a GPU Out-of-Order Scheduling Technique for Stall Reduction.",
  "paper_url": "https://doi.org/10.1109/ISCA59077.2024.00011",
  "abstract": "Graphics Processing Units (GPUs) use massive multi-threading coupl
},
{
  "title": "AVM-BTB: Adaptive and Virtualized Multi-level Branch Target Buffer.",
  "paper_url": "https://doi.org/10.1109/ISCA59077.2024.00012",
  "abstract": "Branch Target Buffer (BTB) plays an important role in modern proce
},
```


Fig. 2. The format of raw data.

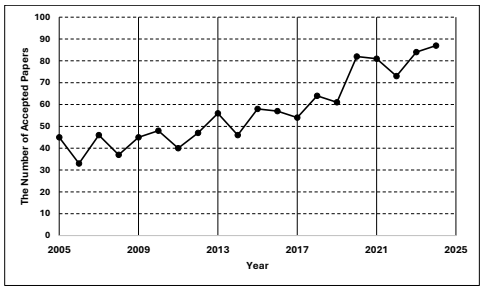

Fig. 3. The trends in the number of accepted papers

	Test	Training
# of Q/A Pairs	**673**	**2187 (Training)** **234 (Validation)**

Table 1. The number of Q/A pairs for test and training.

answer pairs, I used different prompts for each model, due to differences in their capabilities. Since the Mistral model has a large number of parameters, it is known to show GPT-3.5-like performance. Therefore, it is able to make question and answer pairs with a simple prompt. Conversely, because the Gemma model has fewer parameters, it requires more detailed prompt. The differences in the prompts are shown in Figure5.

With this prompt, the LLM models generated training and test datasets. Each data contains four fields: **instruction**, **context**, **question**, and **answer**. The **instruction** field specifies what the LLM model should do during the training and test process: *"Read the abstract and answer the question"*. The **context** field contains the abstract section, based on which the LLM model generates an answer The **question** field contains a question generated by either the Mistral or Gemma model. The **answer** field contains a pre-generated answer which is understood as the ground truth.

The number of test cases is 673 and the number of training sets is 2421. For validation, I used 10% of datasets as a validation data. Therefore, the number of training sets is 2187, and the number of datasets for validation is 234.

2. LLM Selection and Training Details

While selecting the LLM model, the first factor I considered was hardware constraints. LLM models require a large amount of memory to store and process their parameters during training. For this project, since I could only use Google Colab, I chose GPT-2 small model [4] as the

```
"You are a research assistant generating question-answer pairs from scientific paper abstracts.

Title: {title}

Abstract: {abstract}

Generate 2 to 3 QA pairs based only on the abstract. Respond in JSON format:

[
  {{
    "question": "...",
    "answer": "..."
  }},
  ...
]
```

```
"""You are a research assistant.

Given the following abstract of a scientific paper, generate 2 to 3 diverse and
meaningful question-answer (QA) pairs that can be answered solely based on the abstract.

Respond ONLY in valid JSON format with a list of dictionaries.
DO NOT include explanations, notes, or formatting outside the JSON.

Title: {title}

Abstract: {abstract}

Your response:
"""
```

Fig. 4. The difference in prompt. The first prompt is provided to Mistral and second prompt is given to Gemma

baseline. To generate datasets, I used Gemma2-2b-it and Mistral 7b models. The reason for using both models was to observe the results when the training and test datasets are generated by different model. For this reason, I also created test set using Gemma2-2b-it model and made a comparison.

For training, I used two types of training methods. First, I adopted full parameter fine-tuning method. Full parameter fine-tuning method is the same as traditional training method: calculate the loss and modify all parameters. In addition to full parameter fine-tuning, I used LoRA [5] technique. LoRA is a kind of adapter-based fine-tuning. Instead of modifying the parameters of LLM, LoRA creates two small matrices and only updates these matrices during training. The advantage of LoRA is that, because it modifies only a few parameters, it converges very fast. Moreover, since it did not modify any parameter in LLM, LoRA did not degrade performance of the LLM. Therefore, in this project, I compare three models: Baseline GPT-2, Full fine-tuned GPT-2, and GPT-2 with LoRA.

3. Evaluation

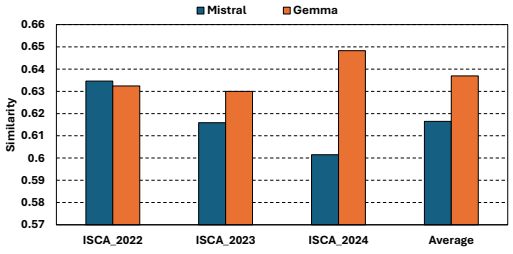

Fig. 5. The accuracy of baseline model

For the baseline model, I used two types of test datasets. One is generated by Mistral model

and another is generated by Gemma model. To evaluate accuracy, I used similarity-based check. For this evaluation, I used the Sentence Transformer [6] to convert the generated sentences into vectors and then computed cosine similarity between them. Therefore, the accuracy means difference between the generated answer and pre-generated answer. Figure 5 shows the result indicating that the average similarity is 61% and 63% for each dataset, respectively.

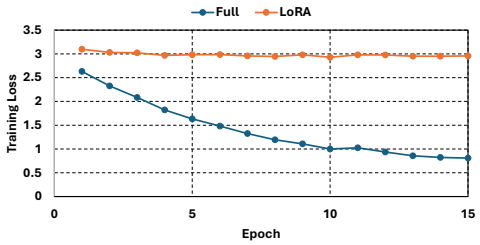

Fig. 6. The accuracy of baseline model

For the fine-tuning methods, I first compared the training time and convergence rate. In terms of training time, fine-tuning with LoRA took 25% less time compared to full parameter fine-tuning. In terms of convergence rate, full parameter fine-tuning shows large decrease during the first few epochs, followed by a steady decrease. Therefore, full parameter fine-tuning is effective at reducing the training loss. On the other hand, fine tuning with LoRA exhibits a very small initial decrease in loss and it converges very quickly - in this case even with 5 epochs. Therefore, although LoRA converges faster, it does not reduce the training loss as much as full-parameter fine-tuning. Overall, in terms of training time and convergence speed, LoRA is more efficient, but in terms of training loss, full parameter fine-tuning is more effective.

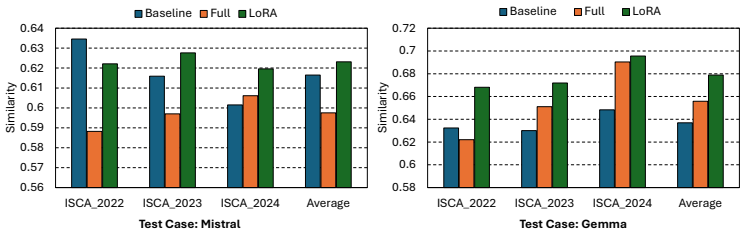

Fig. 7. The similarity after fine-tuning: For the test case generated by mistral, no improvement in similarity is observed; in fact, slight degradation occurred. In contrast, for the test case generated by Gemma, fine-tuning resulted in an improvement in similarity.

First, for the test cases generated by Mistral, the result indicates that my fine-tuning policy was flawed. In the case of full parameter fine-tuning, no improvement was observed. Even it shows slight degradation. I think this consequence comes from loss calculation method I used. For loss calculation, I adopted the traditional calculation method: naively compare the result to the ground truth and compute difference. However, this approach is not suitable for my project. In this project, the criterion should be the difference in meaning rather that ths surface structure of the sentences. Even if two sentences differ in structure, as long as their meanings are the same, the loss should ideally be zero. However, the traditional loss calculation method does not account for semantic similarity, which likely caused the degradation. The results shown in right

graph support this hypothesis. With fine-tuning, the model tends to generate structurally similar sentences, which are more likely to be judged as having high similarity.

Another insight from these results is the effect of LoRA technique. As shown in Figure**??**, LoRA outperforms full-parameter in terms of training time. In addition to the reduced training time, Figure7 shows that LoRA achieves better performance compared to full parameter method. In cases of degradation, LoRA exhibits less degradation, and in cases of improvement, LoRA consistently achieves greater improvements than full-parameter fine-tuning. Taken together, these results indicate that LoRA is superior to full parameter in terms of both training time and performance.

4. Conclusion

There are several issues in my experiments. First, there is a problem with the loss calculation during the training process. During the testing phase, I compared the generated answers and pre-created answers using Sentence BERT and cosine similarity, focusing on semantic meaning. However, during the training phase, I used the traditional loss calculation method, which likely contributed to the suboptimal results. I should align the loss calculation method with the evaluation method used during the testing process. As a next step, to develop a practically fine-tuned LLM model, this issue must be resolved. Otherwise, the LLM may generate sentences that only appear structurally similar to the training data without truly matching in meaning.

Second, I found that the LoRA technique achieves better results in terms of both training efficiency and overall performance. Initially, I expected that LoRA, which modifies only a limited number of parameters, might show worse performance than full-parameter fine-tuning. However, this expectation was incorrect; LoRA demonstrated better performance in this project. This is an important takeaway from the experiment.

References

1. "The International Symposium on Computer Architecture", https://iscaconf.org/," .
2. G. Team, M. Riviere, S. Pathak, *et al.*, "Gemma 2: Improving open language models at a practical size," (2024).
3. "https://huggingface.co/mistralai/mistral-7b-v0.3," .
4. A. Radford, J. Wu, R. Child, *et al.*, "Language models are unsupervised multitask learners," OpenAI blog **1**, 9 (2019).
5. E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," (2021).
6. "https://huggingface.co/sentence-transformers," .