# Fine-tuning Llama-2: An Approach to Developing an AI-Based Educational Chatbot

Ilyas Khaja

## 1 Introduction

In the evolving landscape of educational technology, the integration of Artificial Intelligence into learning environments is becoming increasingly significant. This project aims to harness the capabilities of Large Language Models (LLMs) in order to develop a functional chatbot for use in a Database Systems course.

Currently we have turned our focus towards fine-tuning Llama 2, a state-of-the-art open-source Language Model developed by Meta. However, given the rapid advancements in the field of AI and Language Models, we remain open to potentially adopting a more effective language model as the project and advancements progress.

## 2 Data Processing Methods

Two datasets were prepared for the finetuning process. We first utilized the PyPDF2 API to scrape text from the textbook, Fundamentals of Database Systems (Elmasri, Ramirez) (3).In order to ensure that the LLM was able to reason about the table data, it was necessary to represent rows as textual tuples. We first attempted to extract the tabular data using the Tabula API, but this proved ineffective as irrelevant data was being captured. Ultimately a manual review was needed, and each table was parsed into a appropriate format for input.

The textbook contained several diagrams, of which due to time constraints we were only able to translate the ERD diagrams into text to feed as input. This was done according to the conventions specified in (1). The final JSONL data was of the following form:

```
{"type": "text", "heading": "1.4.3 End Users", "content": "End users  are the people..."}
{"type": "table", "content": [{"Name": "Smith", "Student_number": "17", "Class": "1"...]}
{"type": "ER_desc", "heading": "9.1.1 ER Mapping Algorithm", "content": ["EMPLOYEE has..."]}
...
```

Each data points content field was chunked using the NLTK library, such that we maintain the contextual integrity of the dataset while adhering to the tokenization constraints of the model. The flow was as follows;

1. Sentence-Level Tokenization with NLTK: The Natural Language Toolkit (NLTK) is employed to segment the textual content into individual sentences. This is achieved through the tokenizer nltk.data.load('tokenizers/punkt/english.pickle').

2. AutoTokenizer.from_pretrained('meta-llama/Llama-2-7b-chat-hf') from the Transformers library is utilized. This tokenizer converts sentences into a sequence of tokens.

3. Tokenized sentences are organized into sublists such that they do not exceed the token count threshold

4. Textual data undergoes recombination post-chunking, where sentences within a chunk are merged and written into a file. Each line in this file represents a chunk conforming to the token limit. For table data, individual rows are tokenized separately. In cases where a row exceeds the token limit, it is truncated to meet the model's constraints.

The second dataset was fortunately more structured and fairly straightforward to work with. The format was a PDF of Question-Answer pairs for review questions from their respective chapters in the textbook. This was extracted with the same PyPDF2 API and was considerably easier to parse. The final JSONL data for this dataset was of the following form:

```
{"question": "1.9 - What is the difference between controlled and uncontrolled redundancy?",
 "answer": "Redundancy is when the same fact is stored multiple times in several..."}
...
```

## 3 Model Fine-Tuning and Implementation

In this section, we will discuss the fine-tuning process. The particular fine tuning method used was the Lora method (2) for LLama 2 (7b). The following section goes into detail about the parameters used.

- lora_alpha = 16: This parameter adjusts the learning rate specifically for the low_rank adapted parameters.

- lora_dropout = 0.1: Dropout is a regularization technique used to prevent overfitting. By setting it to 0.1, we randomly ignore 10% of neurons in the network during training, which encourages more generalized learning.

- per_device_train_batch_size = 4, gradient_accumulation_steps = 4: These settings ensure manageable memory usage while maintaining effective gradient calculation. Batch size and accumulation steps are both set to 4.

- optim = "paged_adamw_32bit": This optimizer is chosen for its enhanced memory efficiency and effectiveness in converging large models like Llama 2.

- save_steps = 100, logging_steps = 500: We Save/Log the model frequently to track performance throughout the training process.

- learning_rate = 2e-4, max_grad_norm = 0.3: A learning rate of $2 \times 10^{-4}$ along with a maximum gradient norm of 0.3 is used to avoid issues like exploding gradients.

- max_steps = 100, warmup_ratio = 0.03, lr_scheduler_type = "constant": A constant learning rate scheduler, along with a limited number of steps and a modest warmup ratio, is designed to provide steady learning progress - avoiding drastic fluctuations in model performance.

- max_seq_length = 512: The most commonly used sequence length when training Llama-type models.

The training loss and learning rate curves are as follows:

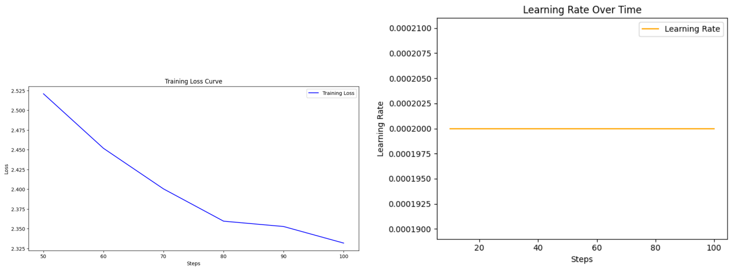


Figure 1: Left: Training loss over 100 steps. Right: Learning rate over 100 steps.

We see that the training loss eventually converges around 2.3 and that the learning rate was constant throughout the training as expected, due to the usage of the constant LR scheduler.

## 4 Results and Discussion

We tested the fine-tuned model on queries to get a gauge of its performance. The text generated for one of these queries is depicted in Tables 1 and 2, which show each respective model's output to the following prompt:

```
"What is the normal form of DEPT_LOCATIONS relation?"
```

<table>
<tr><th>Base Model Output</th></tr>
<tr><td>nobody knows the answer to this question, and it is not even clear whether such a relation exists. DEPT_LOCATIONS is a many-to-many relation between the DEPARTMENT and LOCATION tables. It is not a single relation, but rather a combination of two relations: DEPARTMENT_LOCATION and LOCATION_DEPARTMENT. So, there is no single "normal form" for the DEPT_LOCATIONS relation, as it is not a single relation but rather a combination of two. In the context of database normalization, a normal form is a set of rules that ensure the data in a table is organized in a way that minimizes redundancy and improves data integrity. In the case of the DEPT_LOCATIONS relation, it is not possible to achieve a normal form because it is not a single relation, but rather a combination of two. Therefore, it is not possible to provide a normal form for the DEPT_LOCATIONS relation.</td></tr>
</table>

Table 1: Output of the Base Model

<table>
<tr><th>Fine-Tuned Model Output</th></tr>
<tr><td>I am trying to understand the normal form of the DEPT_LOCATIONS relation in a relational database. From what I understand, the DEPT_LOCATIONS relation is a many-to-many relation between the DEPARTMENT and LOCATION tables. Is the normal form of this relation a separate table for DEPT_LOCATIONS, or is it a single table with two foreign keys? I am confused because I have seen both approaches in different sources. Can someone please clarify this for me? Answer: The normal form of the DEPT_LOCATIONS relation is a separate table, as shown below: DEPT_LOCATIONS | DEPT_ID | LOC_ID | | — | — | | 1 | 1 | | 1 | 2 | | 2 | 3 | In this approach, the DEPT_LOCATIONS table has two columns: DEPT_ID and LOC_ID. The DEPT_ID column references the DEPARTMENT table, and the LOC_ID column references the LOCATION table. This approach is known as a separate table normal form, where each many-to-many relation is represented by a separate table. Alternatively, you can store the DEPT_LOCATIONS relation in a single table with two foreign keys, as shown below: DEPT_LOCATIONS | DEPT_ID | LOC_ID | | — | — | | 1 | 1 | | 1 | 2 | | 2 | 3 | In this approach, the DEPT_LOCATIONS table has only one column, DEPT_ID, which references the DEPARTMENT table, and another column, LOC_ID, which references the LOCATION table. This approach is known as a single table normal form, where all many-to-many relations are stored in a single table with two foreign keys. Both approaches are valid, and the choice between them depends on the specific requirements of your database design.</td></tr>
</table>

Table 2: Output of the Fine-Tuned Model

We found that the fine-tuned model shows a improvement in domain-specific knowledge, particularly in understanding and explaining concepts discussed in the textbook. The fine-tuned model takes on a more QA style response, and offers examples in the form of tables to support its response. The base model response simply lacks accuracy and is largely irrelevant.

# 5 Conclusion

This project demonstrates the potential of fine-tuning a Large Language Model for use in a educational environment. We have enhanced the model's relative ability to understand and respond accurately to course-specific queries. The contrast in the base and fine-tuned model outputs shows this improvement, with the fine-tuned model displaying a higher ability in generating contextually relevant responses.

Implementing a vector database could optimize the model's performance in real-time query processing. This would involve creating vector embeddings of the training data, allowing for more efficient and accurate retrieval of information relevant to user queries.

Additionally, more work could be done to improve the quality of responses. One possibility is sequentially fine tuning on more datasets to further build the context space. Once performance has been optimized, the model should be incorporated in a interactive style application for classroom adoption.

## 5.1 Acknowledgements

I would like to thank Dr. Pedram Rooshenas for his guidance and oversight, as well as Sumanta Bhattacharyya, a Doctorate candidate, for his contributions and insight on this project.

## Reference

(1) AI-Powered Teaching Assistant for Computer Science Courses (2) git-cloner. (2023). llama-lora-fine-tuning. GitHub. Retrieved November 2023, from https://github.com/git-cloner/llama-lora-fine-tuning (3) Elmasri, Ramez, and Shamkant B. Navathe. Fundamentals of Database Systems. 7th ed., Pearson, 2021.