LLMs Project Report

Simon Fliegel
Computer Science (Master) / Secrets Behind LLMs
Matriculation number: 5342785
simon.fliegel@mailbox.tu-dresden.de

Abstract

This document is a supplement to the general instructions for *ACL authors. It contains instructions for using the LaTeX style files for ACL conferences. The document itself conforms to its own specifications, and is therefore an example of what your manuscript should look like. These instructions should be used both for papers submitted for review and for final versions of accepted papers.

1 Introduction

The task was to use the base model *Meta Llama - 8B* and tweak it to generate answers for cultural questions. There are two types of tasks related to these questions: Multiple Choice Question (MCQ) and Single Answer Question (SAQ). For both task two CSV-files, one for training and one for testing file, were provided. The predictions for MCQ and SAQ are written to separate files. These prediction files had to be submitted to the Codabench Competition of the module to participate. After submission the predictions were evaluated and the accuracy was scored for the two tasks. The goal of the project was to experiment with the base model and apply the techniques we learnt throughout the course to a real world problem. The benchmark platform is a nice way to keep track of own improvements and compare them to the other participants in the course. In the next section, I will describe the steps I took to improve the accuracy in detail.

2 System Design and Optimization Steps

2.1 Hello World!

The first step was to gain access to the base model *Meta Llama - 8B* through Huggingface (AI@Meta, 2026). To use the model it can be loaded from Huggingface-API. After successfully loading the model, it can be already used to generate predictions by passing in a prompt and some configuration arguments. Consequently my first running example just received a prompt as user input, passed it to the model, and printed the response back to the console and terminates.

2.2 Base Model Performance

With a running "Hello World" example, I had everything set up to a point where I could start working on the project. My first goal was to test this base model without any tuning on the test datasets to see how it performs and to set a base line for later improvements. I analyzed the format of the test files to extract the necessary information that can be used as input for the model. Firstly, I just extracted the `prompt` field for MCQ and the `en_question` for SAQ and used it as input. Later, I added an additional `instruct` field where I can add context to each prompt for flexibility with e.g., *In-context learning*. The results of the base model on the test files were quite mixed. While it did answer some of the MCQs correctly and also in the correct format it had trouble with SAQs. Often it just repeated the question or asked a related question as answer until it ran out of tokens. As I initially planned to try out fine tuning anyways I didn't spend much time fixing the performance of the base model there and hoped that these problems would vanish once the model is properly tuned to these tasks.

2.3 Fine-Tuning

For fine-tuning I opted for Parameter-Efficient Fine-Tuning (PEFT) using Low-Rank Adaption (LoRA) layers as a very common choice with many online resources available (Sharma, 2025). With LoRA, the base model's weights are frozen and special layers are added to specified *target modules*. Traditionally, fine-tuning updates a weight matrix W of size $d \times d$. However that is demanding and can lead to the base model "forgetting" things it has already

learned. LoRA approximates the change ΔW by multiplaying two smaller matrices A and B of sizes $d \times r$ and $r \times d$ respectively. The parameter r represents the *rank* of the LoRA layer and affects the *influence* of the fine-tuning together with the parameter α. A higher rank means that the weight update is applied to more dimensions offering more capacity for potentially more complex problems but also requires more compute and memory than a lower rank (Unsloth-AI, 2026). The parameter α is often chosen as $\alpha = 2 \cdot r$ and determines how much weight is given to the new LoRA weights. I did a little research on implementation and ended up using a Python library called `unsloth` as it is very popular, well documented and easy to use and offers great performance and optimizations in that field (CHAWLA, 2025). I did experiment a little bit with `rank`, `alpha` as well as the `target modules` which define where the LoRA layers are added. Initially I used $r = 16$ and the target modules `q_proj`, `k_proj`, `v_proj`, and `o_proj` representing the four attention modules query-, key-, value-, and output-projection. Later I increased the rank to $r = 64$ and added `gate_proj`, `up_proj`, `down_proj`, and `lm_head` to target modules affecting intermediate representations and also the final output layer. While this increased the resource requirements it didn't really affect the results so I stopped spending time on parameter optmization and tried other things.

2.4 Combining Tasks

Right from the beginning I asked myself whether it was necessary to train one model for each task or if a single model can be trained for both leveraging the knowledge it leant in the training data of the one task and applying it to questions of the other. When using a single model, the task has to be encoded somehow in the prompt so the model knows what answer format is expected. However, with the additional instruction field I added earlier, this was definitely possible.

2.5 Data Augmentation

While it felt like I was in a dead-end with hyperparameter tuning and task separation and my score was pushed down more and more in the leaderboard I revised the lectures to see what other options I had to improve the results. The one thing where I haven't spent much thought on was the data itself. I thought about synthetic data generation and using a bigger model to generate additional training data. However, I wasn't sure whether the quality of the generated data would be good enough as I could not validate it myself, also it would have cost me significantly more time to set this up and also it felt a bit like cheating exploiting a bigger model to generate more training data. When first writing the code for extracting the data from the `train_dataset_saq.csv` I only took the answer with the highest count as being the answer with the highest certainty of correctness. However, to gain more training data I changed this to multiplying the question and using each possible answer. Now the question was whether something similar was possible to the MCQ data. Actually I haven't used all information there either. For each answer option there was given the name of that country for which this answer would be true. I didn't give this much thought in the beginning either but this allows to increase the number of training samples for MCQ by four times using each answer option as a separate training sample. The only thing that has to be done for that is replacing the country in the initial question with the one from the answer option and changing the correct answer in the training data to that option.

3 Results

3.1 Improvements

3.2 Conclusion

4 Tools

Throughout working on this project, *Google Gemini Pro* was used for coding asssistance and fixing grammar and typos in this report.

References

AI@Meta. 2026. Llama-3-8b on hugging face.

AVI CHAWLA. 2025. Top 4 llm fine-tuning frameworks!

Sanjana Sharma. 2025. What is parameter-efficient fine-tuning (peft)?

Unsloth-AI. 2026. Unsloth ai - fine-tuning llms guide.

A Example Appendix

This is an appendix.