ECE-GY 7123 / CS-GY 6953 / Deep Learning - Fall '25
Midterm Project Report
Team: Spline Reticulator

Thanh Do
New York University
`qd2121@nyu.edu`

Abstract

This document serves as a midterm project report. It details the introduction, dataset description, model overview, experimentation, results, and conclusions for the Kaggle Math Question Answer Verification Competition. The project involved Supervised Fine-Tuning (SFT) of a Llama-3.1 8B model to predict the correctness of answers to math questions. By iterating on prompt formats, LoRA hyperparameters, and training duration, we developed a model that achieved a final validation accuracy of 89.2%.

1 Introduction

Our midterm project was a Math Question Answer Verification Competition hosted on Kaggle. Participants were tasked with Supervised-Fine Tuning (SFT) of the Llama-3.1 8B model to predict the correctness of answers to math questions. The goal was to determine whether the provided answer to each question was correct or not. Submissions required a CSV file specifying whether the solution for each test set question was labeled as 'True' or 'False', which makes our model effectively a binary classifier.

Due to the known weaknesses of LLMs in multi-step logical reasoning, this project is highly relevant. While powerful, LLMs often "hallucinate" or produce plausible-sounding solutions that contain critical calculation or logical errors. This makes them unreliable for high-stakes domains like education or STEM research. This project requires us to create a "verifier" or "evaluator" model, a critical component that can filter these incorrect outputs, improve the reliability of LLM-generated content, and act as a check against model hallucinations.

2 Hypothesis & Verification

2.1 Understanding LoRA Fine-Tuning

Low-Rank Adaptation (LoRA) (Hu et al., 2021) is a parameter-efficient fine-tuning (PEFT) method designed to adapt large language models to specific tasks without updating all model parameters. By introducing trainable low-rank matrices into specific layers of the Transformer architecture, LoRA significantly reduces the number of trainable parameters, leading to faster training and lower computational costs.

A popular extension of this is QLoRA (Quantized LoRA) (Dettmers et al., 2023), which achieves even greater memory savings by loading the base model in a compressed, 4-bit quantized format.

For this project, we utilized both standard LoRA (running in 16-bit precision) and QLoRA (which combines LoRA with 4-bit quantization) to experiment with the trade-offs between training speed and memory efficiency.

2.2 Hypothesis

Our main hypothesis is that a Llama-3.1 8B model's ability to verify mathematical reasoning is not primarily a function of its raw parameters, but is unlocked by a highly specialized fine-tuning (SFT) strategy.

Specifically, we hypothesize: "Fine-tuning a Llama-3.1 8B model using LoRA on a dataset that explicitly includes step-by-step natural language reasoning (as provided in the `solution` column of the dataset) will result in a classifier with significantly higher accuracy than a model trained naively. Furthermore, this accuracy will scale with the size and quality of this reasoning-focused dataset."

2.3 Verification Strategy

Our primary metric is *classification accuracy*: the percentage of correct "True"/"False" predictions on a held-out test set. Our findings (Table 1) show a clear progression from a baseline = 0.61 accuracy to 0.892, which directly supports our data-centric hypothesis.

We will also use the ranking on the private Kaggle leaderboard to validate our approach.

3 Dataset Description

3.1 Dataset

The Kaggle competition requires us to use the `ad6398/nyu-dl-teach-maths-comp` dataset hosted on Hugging Face. The task was to train a model to predict the `is_correct` boolean label using the provided question, answer, and solution.

Each row in the dataset contains four primary fields:

- `question` – The math question posed.
- `answer` – The ideal or correct answer to the question.
- `solution` – A detailed reasoning or solution that explains the answer.
- `is_correct` – The target label (`True`/`False`).

The complete training dataset contains 1,000,000 samples. In our training pipeline, we used a fixed random seed to shuffle the data and create a validation set of the last 500 samples after shuffling, with the remaining samples forming our internal training dataset.

3.2 Formatting

For training and inference, all data was formatted into an unified prompt, as detailed in Figure 2. Some of our experiments (e.g., Section 7.4) use a slightly different prompt; deviations are clearly noted in the corresponding experiments.

3.3 Preprocessing

We perform some light preprocessing in our final model. Please refer to Section 2 for more background and details.

4 Model Overview

Base Model: The base model that we use in this task is the `unsloth/Meta-Llama-3.1-8B` model. We started from the starter notebook & utilized the Unsloth library's `FastLanguageModel` for optimized training. The Unsloth implementation includes custom kernels for faster performance.

LoRA Configuration: We initially experimented with QLoRA (Dettmers et al., 2023), then trained our final model and ran inference on it using full 16-bit LoRA (Hu et al., 2021). We targeted both the Attention and MLP layers, as recommended in Schulman and Lab (2025). A global seed of 313337 was used for reproducibility.

We tried to apply a few different hyperparameter values in our experiments. The following LoRA hyperparameters were used for our final model:

- **LoRA Rank (r):** 32
- `lora_alpha`: 64
- `lora_dropout`: 0

Training Strategy:

We used the `SFTTrainer` class from the Hugging Face `TRL` library. Training parameters are as follows:

- **Batch Size:** 32
- **Gradient Accumulation Steps:** 1
- **Optimizer:** `adamw_torch`
- **Max sequence length:** 2048
- **Max training steps:** 7500
- **Warmup steps:** 5
- **Weight decay:** 0.01
- **Learning Rate:** 2×10^{-4}
- **LR Scheduler:** `linear`

Please refer to Figure 2 for the training and inference prompt.

About the **number of epochs**, due to the limitations of time and computing resources, we did not make use of the whole training dataset; thus, effectively, it is **less than one full epoch**. The training process is limited by the **max training steps** hyperparameter as specified above.

The training environment is detailed in Section 5.1.

Evaluation: We evaluate the performance of our models by measuring *classification accuracy* on a common validation dataset splitted from the training set as specified in Section 3.1. An overview of the result for our experiments can be found in Table 1.

question: What is the radius of the circle inscribed in triangle (ABC) if $(AB = 22)$, $(AC = 12)$, and $(BC = 14)$? Express your answer in simplest radical form.

answer: 3.16227766016838

solution: The circle is inscribed in a triangle, and we know the sides of the triangle. To use the inradius formula, we need the area of the triangle. We can use Heron's formula to calculate the area.

```
import math
from sympy import *

AB, AC, BC = 22, 12, 14
# Calculate the semiperimeter and area using Heron's formula
s = (AB + AC + BC) / 2
K = sqrt(s * (s - AB) * (s - AC) * (s - BC))
print(K)  # 75.8946638440411

# Now use the formula for the radius of the inscribed circle
r = K / s
print(r)  # 3.16227766016838
```

Using the inradius formula, the answer is (3.16227766016838).

is_correct: True

Figure 1: An example question & answer row from the competition dataset.

```
You are an expert mathematician and a meticulous verifier.
Your task is to evaluate a proposed solution to a math problem
and determine if it's correct or not.
Carefully read the Question and the Solution. Determine if the
Solution is a correct reasoning process to solve the Question.
Your response should be 'True' if the solution is correct,
otherwise 'False'.
Below is the Question and Solution.
Question:
{}
Solution:
{}
Output:
{}
```

Figure 2: The prompt template that we used for both SFT training and inference. The placeholders {} are filled with the question, solution, and a ground-truth label for each training example. During inference, the output label is left blank for the model to fill in.

5 Methodology and Experimentation

5.1 Environment Details

The training and inference process for our model was conducted on Google Colab using an A100 GPU with 40GB of VRAM and 80GB of RAM. We only utilized the 100 free monthly compute credits provided to us, which translated to approximately 17 hours of compute on the specified runtime type. The version of the Colab notebook runtime environment was the latest as of this writing (`2025.10`).

5.2 Establishing Baseline & Initial Approach

To establish a baseline for our later evaluations, we started with the starter Jupyter notebook, patched the dependency setup cell, populated the same training and validation dataset that we'll use for all experiments as specified in Section 3.1, then used the starter model with all parameters unchanged (including the prompts) to get a validation accuracy value. For the starter model training process, we kept the original seed value of 42.

With the above setup, we obtained a baseline accuracy of **61%**, which is somewhat decent but not enough to get a full point for this assignment section.

From reading the implementation, we can see that some parameters were intentionally set to low just to demonstrate the concept. We were able to figure out a few initial ideas that would be useful for improvement:

- **Change the LoRA rank r from 1 to 16 (and α to 32 accordingly)**.This is a recommended value in various guides, which allows our model to capture more complex relationships from the input.
- **Change the maximum training steps from 60 to 400.** Contrary to what's stated in the starter notebook, the training process there did not run for a full epoch yet; only `n_steps * effective_batch_size` $= 60 \times 2 \times 4 = 480$ samples from the training dataset were used. By running the training process slightly longer, the model would have been exposed to more data, allowing it to generalize better while keeping the training time reasonable for quick iterations (around 10 minutes to run all cells on the A100).
- **Change the maximum sequence length to 2048.** While running the validation process, we encountered cases where PyTorch complained that the tokenized input was longer than the sequence size. This can be problematic during training, as the judgment label (True/False) might be truncated. While the training and inference time can be longer as a result of this change, it should improve the overall accuracy of our model.

With those simple changes, the model achieved **77%** accuracy on the validation set. While this is a significant improvement, it is still not sufficient to score full points for this section, so we decided to explore other ways to enhance the model's performance.

5.3 First Attempt: Prompt Engineering & Data Cleaning

A very popular and powerful technique when making large language models more useful is **prompt engineering**, so we decided to explore this approach. From the starting prompt, we made the following modifications:

- We would like to emphasize the *reasoning verification* behavior of the model, so we added it to the prompt ("You are an expert mathematician **and a meticulous verifier**...");
- We would also like our model to focus especially on the Solution section of the input and its relation to the Question part, so we have added it to the prompt.

Another possible venue of improvement is **data cleaning**. As we can see from the example question (Figure 1), decimal numbers are represented with very high precision in the samples. Our line of reasoning is that since we are working with *language models*, which do not directly deal with the *numerical value* of those tokens, retaining the high-precision numbers as they are might dilute the model. Therefore, we implemented a quick data filter to detect and truncate these values to only 4 decimal places for both training and inference.

After implementing the proposed changes above with some minor tweaks (more details can be found in Section 7.4), our model achieved a validation accuracy of **79.4%**, which is lower than we initially expected. It's possible that we are not very good at optimizing prompts, so we decided to continue with tweaking other hyperparameters instead.

5.4 Second Attempt: Hyperparameter Tweaking & Training Optimization

In this attempt, we aimed to enhance the model's performance by exposing it to more data, potentially improving its generalization capabilities.

We retained the modified prompt and the data cleaning step from our previous phase. Our iterative experiments during this phase included:

- **Further Increasing LoRA Rank:** We tested a more expressive adapter by increasing the rank from $r = 16$ to $r = 32$ and `lora_alpha` from 32 to 64.
- **Increasing Training Steps:** We progressively increased the training duration, evaluating performance at checkpoints corresponding to 1,000, 7,500, and 8,500 steps.

These proposed changes significantly lengthened the required training time; given our limitations in time and computing resources, training optimization became even more important. For this aspect, we implemented the following approaches:

- **Switching to 16-bit LoRA:** During our previous training, we noticed that we're not making full use of our available VRAM. For this final, most intensive training run, we disabled 4-bit quantization (`4bit = False`) to prioritize faster training speed over VRAM usage. Since we are using full 16-bit LoRA, the optimizer needs to be changed to `adamw_torch` to prevent unnecessary quantization during the optimization step.
- **Increasing per-device train batch size to 32 and decreasing gradient accumulation steps to 1:** With more VRAM, we can load more samples at once & further parallelize the training process. By increasing the per-device training batch size, we reduced the number of transfers required between RAM and VRAM for each training step, thereby accelerating the training process.
- **Setting `dataloader_num_workers = 8`:** Theoretically, this should further parallelize the data transfer process from RAM to VRAM between each step, but we did not observe any big differences during training with this option.

With this approach implemented, our model achieved a validation accuracy of **82.8%** after 1,000 training steps in the first evaluation stage, indicating that we are heading in the right direction. As such, we left the training process to run for another 11 hours, then performed further evaluation on the 7,500- and 8,500-step milestones. At the end of our training run, the model yielded a nice **89.2%** validation accuracy, so we decided to go with this model for our submission.

6 Result

We submitted results from two models to Kaggle for evaluation: one trained for 7,500 steps, and the other for 8,500 steps. After the competition, we were pleased to see that our entries achieved a final score of `0.86120`, **placing our team** 12th **out of 71 participating teams.**

7 Analysis

7.1 Performance Comparison

The model's performance (Validation Accuracy) was evaluated before and after retraining iterations (Table 1). A clear improvement was observed as the model's rank and training duration were increased.

7.2 Training Loss Analysis

We logged the training loss at 10-step intervals to monitor convergence. The loss curve showed a consistent downward trend, starting from 1.086200 at step 10 and steadily decreasing to 0.330500 by step 8460, indicating the model was effectively learning from the data.

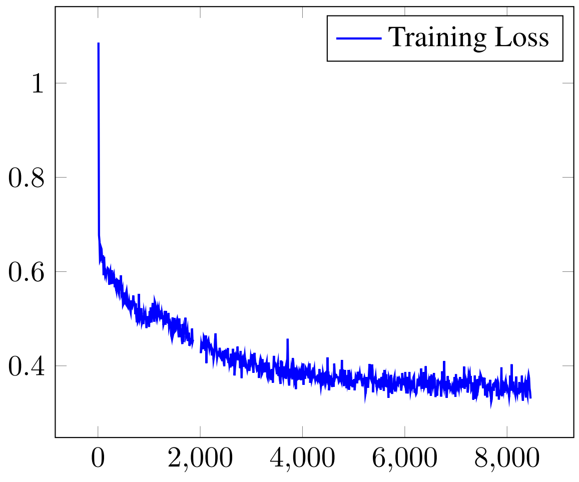

Figure 3: Training loss curve for the phase 2 model (r=32, 8k steps). The gap in the line corresponds to a missing range in the logs (steps 1860-2010) due to a technical difficulty.

The optimization process seems to have plateaued around the 5,000-step mark. It would be interesting to see if this trend continues when training our model with the full dataset and for a greater number of epochs.

7.3 Factors Contributing to Improved Scores

The observed improvement in scores after retraining can be attributed to several factors:

Increased Data Exposure: Increasing the number of training steps from 60 to 400, then to over 8,000, provided the model with more examples, thereby enhancing its ability to generalize to unseen data. This is especially true in this dataset since some questions contain very specific math knowledge that can't be generalized to other questions (e.g., probability, combinatorics, geometry, ...).

Hyperparameter Tuning: Increasing the LoRA rank r from 16 to 32 allowed the adapter to learn more complex patterns, which was crucial for this reasoning task.

Full-Precision Adapters: Switching from 4-bit QLoRA to 16-bit LoRA for the final run dedicated more computational resources, thereby removing

Experiment Configuration	Validation Accuracy
Baseline (Section 5.2)	0.61
QLoRA, $r = 16$, 400 steps (Section 5.2)	0.77
QLoRA, $r = 16$, 400 steps, modified prompt (Section 5.3)	0.794
LoRA, $r = 32$, 1,000 steps, modified prompt (Section 5.4)	0.828
LoRA, $r = 32$, 7,500 steps, modified prompt (Section 5.4)	0.882
LoRA, $r = 32$, 8,500 steps, modified prompt (Section 5.4)	0.892

Table 1: Model performance comparison between successive experiment configurations.

any potential performance constraints introduced by quantization.

7.4 Other experiments tried and their results

Besides the main iterative training, we conducted several other experiments:

- **Slightly different prompts during training and inference:** During our initial exploration (Section 5.3), we tried to diverge the training and inference prompts slightly to see what would happen. Using the same prompt as in Figure 2 for both training and inference yielded a validation accuracy of 0.772. However, when we remove the "Carefully read the Question..." line from the inference prompt, the accuracy improved slightly to 0.794. Nonetheless, when testing the same method against a 1000-step-trained model in our Phase 2 experiment (Section 5.4), the improvement was much more modest (0.828 to 0.83). While this is an interesting phenomenon, as it reduces the predictability of our training process without yielding significant gains, we decided not to investigate it further.

- **QLoRA vs. LoRA Speed:** While it's trivial to expect 16-bit LoRA to be faster than QLoRA due to the reduced quantization overhead (some resources claimed the difference to be 66%[1], to confirm this, we ran a brief 100-step experiment comparing the speed of QLoRA (4-bit) vs. standard LoRA (16-bit) while keeping other hyperparameters the same. The performance was very similar, with QLoRA (4-bit) achieving 5.61 samples/sec and standard LoRA (16-bit) achieving 5.76 samples/sec, which is around a 2% difference. This was somewhat surprising; we assumed that Unsloth applied some optimizations to speed up QLoRA. However, since our training dataset is very large, this efficiency gain is still valuable, so we decided to proceed with full 16-bit LoRA for our model.

[1]https://docs.cloud.google.com/vertex-ai/generative-ai/docs/model-garden/lora-qlora

8 Discussion

Our findings so far supports our hypothesis that fine-tuning our model against the given dataset with step-by-step reasoning written in natural language will result in a reasoning classifier, and the accuracy of our classifier scales with the size of the training dataset.

Challenges. While we talked about the *quality* of this dataset, due to the large size of it, we were unable to measure the quality aspect of our hypothesis. Also, due to limited computing resources, we are also unable to extend our training process to the whole available dataset. Another point that failed short in our experiments was the effectiveness of a pure prompt engineering approach, due to our inexperience with this technique.

Future Work. While working on the challenge, we also wondered on how to change the task from a simple True/False classifier to a *reasoning verifier*. A possible approach could be changing the training labels for False examples to be an explanation of the error (e.g., "False. The solution incorrectly adds the discount instead of subtracting it."). This would fine-tune the model to "show its work" and provide genuine, step-by-step verification, which is a far more powerful and useful tool.

9 Conclusion

The experiment demonstrated the effectiveness of LoRA fine-tuning in adapting large language models, such as Llama-3.1 8B, to the specific classification task of math answer verification. Fine-tuning the model with a modified prompt, an increased adapter rank, more training steps, and using 16-bit precision for the final run resulted in a performance improvement from 0.61 to 0.892 accuracy.

Our takeaways include:

- QLoRA provides a resource-efficient and highly effective approach for initial experimentation. However, if resources allow, LoRA provides faster training speed, especially on larger datasets.
- Training duration and data exposure (i.e., number of steps) are critical factors for improving generalization on reasoning tasks.
- Prompt engineering is not just for inference; the structure of the training prompt itself is a crucial hyperparameter for SFT, as it defines the task the model learns.

10 Links to Resources

The corresponding resources for this report can be found at:

Training Notebook:

- Colab: `https://colab.research.google.com/drive/1672B25L9u0f3muTx472tKp_dEkXwcOjr?usp=sharing`
- GitHub: `https://github.com/chitoge/qd2121_dl_midterm/blob/main/training_notebook.ipynb`

Reproducer Notebook:

- Colab: `https://colab.research.google.com/drive/1Wdn07WsQDpbGtv_QuXlo7DHX19-76p5e?usp=sharing`
- GitHub: `https://github.com/chitoge/qd2121_dl_midterm/blob/main/repro_notebook.ipynb`

Model Weights:

- 7,500-step checkpoint (which generates the best-performing submission for our team on Kaggle): `https://drive.google.com/drive/folders/1gVGbqSK5QzQWraf7tM9eIvV9GJ92Z9iF?usp=drive_link`
- 8,500-step checkpoint: `https://drive.google.com/drive/folders/19XThtpoI59avLT7XcWi0Yl4SMP0fQVQb?usp=drive_link`

GitHub Repo: `https://github.com/chitoge/qd2121_dl_midterm`

References

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Preprint*, arXiv:2305.14314.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

John Schulman and Thinking Machines Lab. 2025. Lora without regret. *Thinking Machines Lab: Connectionism*. Https://thinkingmachines.ai/blog/lora/.