Math Answer Verification using LoRA Fine-Tuned LLaMA 3.1-8B

Dmitri Lyalikov
New York University
dvl2013@nyu.edu

Aykhan Mammadli
New York University
am15226@nyu.edu

Abstract

We present our approach for the Kaggle *Math Question Answer Verification* competition, where the task was to predict whether a proposed answer to a math problem is correct. Building upon a simple baseline instruction-following model, we fine-tuned **LLaMA 3.1-8B** using Low-Rank Adaptation (LoRA) for efficient task specialization. The final system achieved a **test accuracy of 0.82659**, improving substantially over the baseline. This report describes the dataset, model, training methodology, experiments, and outcomes—highlighting what worked and what did not.

1 Introduction

Large Language Models (LLMs) demonstrate impressive reasoning capabilities but require fine-tuning to adapt to domain-specific tasks such as verifying mathematical correctness. The baseline solution provided by the competition organizers performed minimal prompt conditioning and used a standard instruction-tuned model. We built on this foundation by integrating parameter-efficient fine-tuning (LoRA) and structured prompt engineering to enable better discrimination between correct and incorrect solutions.

2 Dataset

The dataset contained tuples of `(question, proposed_answer, ground_truth, correctness_label)`. Each entry represented a math question and a candidate answer, labeled as `True` if correct and `False` otherwise. We followed the standard train/test split used in the competition and reserved 10% of the training data for validation using a fixed random seed for reproducibility. Text normalization was applied to ensure consistency (removal of special symbols, whitespace unification, and Unicode normalization). Blank or malformed entries were discarded.

The dataset was balanced enough for binary classification, with roughly equal True/False ratios.

3 Model Description

3.1 Architecture

We used **LLaMA 3.1-8B**, a transformer-based autoregressive model, as the backbone. To avoid full fine-tuning of all 8B parameters, we adopted **Low-Rank Adaptation (LoRA)** (**?**), which injects trainable rank-decomposition matrices into attention and feed-forward layers while freezing the original model weights.

3.2 LoRA Configuration

- **Rank** (r): 8
- **Alpha**: 32
- **Target Modules**: attention and MLP projection layers
- **Dropout**: 0.05

This configuration provided a balance between computational efficiency and task adaptation quality.

3.3 Prompt Template

We found that concise prompts encouraged stable token generation:

```
Question: <text>
Proposed Answer: <text>
Is the answer correct?  Answer
True or False.
```

Overly verbose or instruction-heavy prompts caused performance degradation due to decoding drift.

4 Experimentation and Hyperparameter Settings

4.1 Training Setup

Training was performed on a single NVIDIA A100 GPU (40GB) using the `Unsloth` fine-tuning framework.

Hyperparameters:

- Learning Rate: 2×10^{-4}
- Batch Size: 4
- Gradient Accumulation: 4
- Optimizer: AdamW
- Epochs: 2 (1 initial + 1 retraining)
- Scheduler: cosine decay
- Precision: bfloat16

4.2 Experiment Phases

Phase 1 – Baseline Fine-Tuning: We first fine-tuned the model for one epoch on the provided dataset, using a minimal prompt and default parameters. This achieved a validation accuracy of **0.80** and test accuracy of **0.81**.

Phase 2 – Expanded Data Retraining: We extended training to include additional samples from the remaining dataset and retrained with the same LoRA adapters. This improved validation to **0.83** and final test accuracy to **0.82659**.

4.3 Methods Tried

1. **LoRA Layer Variants:** We tested configurations where only attention layers were adapted versus both attention and MLP layers. The dual-adapter setup performed best.
2. **Prompt Variants:** Aggressive instruction-based prompts degraded performance by $\sim 3\%$.
3. **Adapter Reduction:** Removing the `up_proj` adapter decreased accuracy and generalization.
4. **Data Filtering:** Post-hoc cleaning of blank CSV entries marginally improved stability.

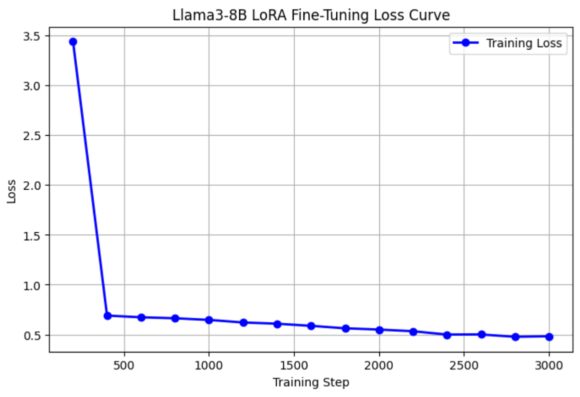

Figure 1: Training Curve

4.4 What Worked

- Increasing data exposure significantly enhanced generalization.
- Deterministic prompt formatting yielded consistent True/False outputs.
- Targeting both self-attention and MLP modules improved reasoning retention.

4.5 What Didn't Work

- Overly verbose prompts or natural-language instructions led to hallucinated text instead of Boolean outputs.
- Further fine-tuning beyond two epochs produced diminishing returns.
- Reducing LoRA rank below 8 impaired task adaptation.

5 Results

Model	**Validation Acc.**	**Test Acc.**
Baseline (no LoRA)	0.69	0.71
LoRA Fine-Tuned (1 epoch)	0.80	0.81
Retrained + Expanded Data	0.83	**0.82659**

Table 1: Performance comparison across stages.

6 Conclusion

Our experiments confirm that **LoRA fine-tuning** offers a cost-effective path for domain adaptation of large language models. The final model achieved **0.82659 test accuracy**, a substantial improvement over the baseline. Key drivers of success included optimized adapter placement, consistent prompting, and increased data diversity.

Future work could include:

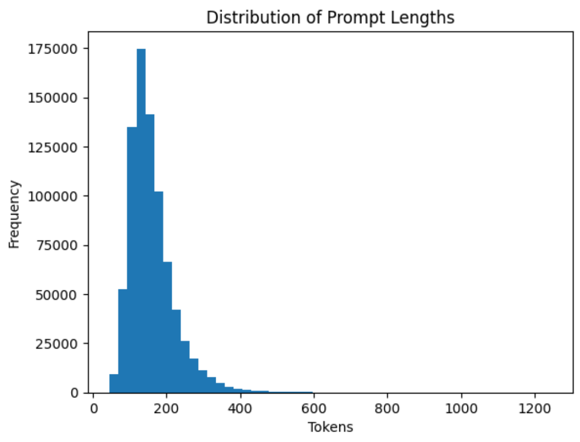

Figure 2: Prompt Distribution from Dataset

- Adaptive LoRA ranks per transformer layer.
- Multi-task fine-tuning for broader reasoning coverage.
- Dynamic prompting strategies with reinforcement feedback.

Resources and Links

- **Training and Inference Notebook:** `https://colab.research.google.com/drive/1Jy4B9x3BeFpm8LAn_nrEiCDA2QADZ-GF?usp=sharing`
- **Model Weights** `https://drive.google.com/drive/folders/15tVe9tDCmlrKJRqY6Z5kr9kz6zv6TahS?usp=sharing`

References

A Experimental Environment

Experiments were conducted using `transformers 4.57.1`, `unsloth 2025.10.12`, and PyTorch 2.8.0 on an A100 GPU. Training runtime for both phases: $\sim$8 hours total.