Comparative Study of Optimizers for Math Fine-Tuning on Qwen3-0.6B

Chenhe Yang 225040201 | Guanyu Mu 225040206 | Xin Lu 225040215 | Fengxiao Qian 225040321

1. Introduction

This experiment aims to fine-tune the Qwen3-0.6B-Base language model on the Math500 dataset to enhance its mathematical problem-solving abilities. We explore the impact of various optimization strategies—specifically comparing SGD, AdamW, and LoRA—on both training dynamics and final evaluation performance, with a particular focus on effectiveness under limited computational resources.

2. Experiment Design

2.1. Objective

This experiment aims to fine-tune the **Qwen3-0.6B-Base** language model on the **Math500** dataset to enhance its mathematical problem-solving abilities. We explore the impact of various optimization strategies and hyperparameter choices on both training dynamics and final evaluation performance. Our goal is to investigate which optimizer—SGD, Adam, or LoRA—offers the most stable and effective training for math reasoning tasks, especially under limited computational resources.

2.2. Dataset

We fine-tuned on the **Math500** dataset using a 90/10% train-validation split. The preprocessing pipeline involved three key steps:

- **Instruction Augmentation:** To guide reasoning, we appended the following chain-of-thought directive to every user prompt:

  "Please reason step by step, and put your final answer within `\boxed{}`*."*

- **Tokenization and Masking:** We utilized the Qwen3 tokenizer and applied **loss masking** by assigning a label value of -100 to all user tokens, excluding them from loss calculation.
- **Filtering:** Input sequences exceeding 512 tokens were discarded to ensure training stability.

2.3. Training

We trained the model using three types of optimizers:

- **AdamW**: A commonly used adaptive optimizer for transformer models.
- **SGD**: A momentum-based stochastic gradient descent method.
- **LoRA**: A parameter-efficient fine-tuning method where we keep the original weights W_0 fixed and only update two small low-rank matrices (B and A). We use Adam as the optimizer for this inner update step.

We experimented with various hyperparameter configurations, including learning rates (1e$-$6, 2e$-$5, 5e$-$5, 1e$-$4), number of epochs (2 or 3), and LoRA ranks (e.g., 8, 16). Each training configuration saved outputs—including the model checkpoints, training/validation loss curves, and logs—to separate folders such as:

```
adam_lr1.0e-06_ep3/
adam_lr2.0e-05_ep2/
lora_8_lr1.0e-04_ep2/
lora_16_lr5.0e-05_ep3/
sgd_lr1.0e-05_ep2/
sgd_lr1.0e-05_ep3/
```

Each output directory contains the model weights, 'training-loss.png', 'val-loss.png', and optimizer logs.

2.4. Training Monitoring and Logging

During training, we recorded and plotted:

- **Training Loss Curve:** Showing how the model's loss decreases across iterations.
- **Validation Loss Curve:** Used to detect overfitting and identify optimal epochs.
- **Resource Usage:** Peak memory usage and training time per configuration.

These plots are used to compare training stability and convergence across different optimizers.

Contributions:
Chenhe Yang responsible for experimental evaluation and comparative analysis;
Xin Lu and Guanyu Mu responsible for use different configurations for model fine-tuning training;
FengXiao Qian responsible for data loading and preprocessing

2.5. Evaluation

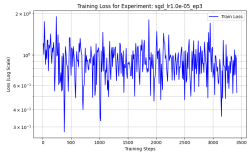
After training, we evaluated all finetuned models on the Math500 test set (500 samples) using exact match accuracy. Model predictions were generated via ‘rollout.py‘ and scored via ‘evaluate.py‘, using the same boxed-answer format as the training prompt. We compare each finetuned model’s test accuracy against the base model without any fine-tuning.

3. Results and Analysis

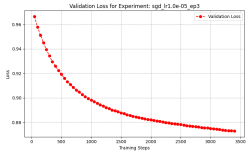
In this section, we present the evaluation results of the finetuned Qwen3-0.6B models on the Math500 test set. We analyze the performance differences across optimizers (SGD, Adam, LoRA), the impact of inference hyperparameters (temperature, max output tokens), and the computational resource trade-offs.

3.1. Training Dynamics and Resource Analysis

To understand the convergence behavior and computational efficiency of different optimizers, we analyzed representative training runs. Figure 1 illustrates the training and validation loss curves, while Table 1 summarizes the computational resource consumption.

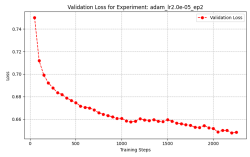


(a) SGD (Epoch 3) - Train Loss

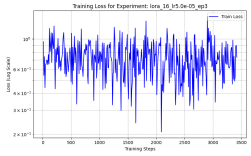


(b) SGD (Epoch 3) - Val Loss

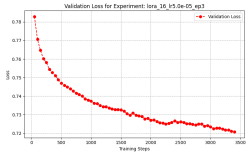
(c) AdamW (LR 2e-5) - Train Loss



(d) AdamW (LR 2e-5) - Val Loss



(e) LoRA (Rank 16) - Train Loss



(f) LoRA (Rank 16) - Val Loss

Figure 1. **Comparative Training Dynamics.** (a-b) **SGD** shows the smoothest convergence. (c-d) **AdamW** exhibits faster initial learning but potential instability. (e-f) **LoRA** is characterized by high variance (noise) in training loss. See Appendix A for more plots.

As shown in Figure 1, SGD (Top Row) maintains a steady downward trend in validation loss without signs of overfitting, proving its robustness. In contrast, LoRA (Bottom Row) exhibits significantly noisier training loss, which is expected as the optimizer navigates a constrained optimization landscape updating only low-rank matrices.

In addition to convergence stability, we evaluated the resource efficiency of each method. Table 1 details the peak GPU memory usage and total training time for various configurations.

Table 1. Computational Resource Usage Comparison

Configuration	Peak Memory	Training Time
SGD (LR 1e-5, Ep 3)	9.0 GB	42 min
SGD (LR 1e-5, Ep 2)	6.8 GB	27 min
AdamW (LR 1e-6, Ep 3)	9.0 GB	45 min
AdamW (LR 2e-5, Ep 2)	9.0 GB	26 min
LoRA (Rank 8, Ep 2)	**5.5 GB**	30 min
LoRA (Rank 16, Ep 3)	**5.6 GB**	51 min

Resource Efficiency Analysis: The resource data highlights the significant memory advantage of Parameter-Efficient Fine-Tuning (PEFT).

- **Memory Footprint:** LoRA configurations consistently required significantly less VRAM ($\sim$5.6 GB) compared to full fine-tuning with AdamW or SGD (up to 9.0 GB). This reduction of approximately 38% allows the model to be trained on consumer-grade GPUs with limited memory.
- **Training Time:** Interestingly, while LoRA saves memory, it does not strictly reduce training time (e.g., LoRA Rank 16 took 51 minutes). This is likely due to the additional computational overhead of the adapter layers during the forward pass, despite updating fewer parameters.

3.2. Performance Comparison of Optimizers

We evaluated the models using a near-greedy decoding strategy ($T = 0.1$) to minimize randomness and prioritize precise reasoning. Table 2 summarizes the peak exact match accuracy for each optimizer.

Consistent with our previous observations, the model fine-tuned with SGD demonstrated superior performance, achieving the highest overall accuracy of **46.40%** (Epoch 2). In contrast, the adaptive optimizer AdamW peaked at 31.20%, and the parameter-efficient LoRA method reached a maximum of 24.00% (Rank 8).

This result reinforces the finding that for this specific model scale and dataset, the momentum-based SGD finds a more robust solution. Notably, the gap between SGD and Adam remains significant even at the optimal temperature setting.

Table 2. Peak Test Accuracy on Math500 (Temperature=0.1)

Optimizer	Configuration	Accuracy (%)
SGD	Epoch 2	**46.40%**
SGD	Epoch 3	44.00%
AdamW	Config A (lr 1e-6)	31.20%
AdamW	Config B (lr 2e-5)	23.40%
LoRA	Rank 8, Epoch 2	23.60%
LoRA	Rank 16, Epoch 3	20.20%

3.3. Impact of Inference Temperature

We investigated the effect of sampling temperature ($T \in \{0.1, 0.3, 0.7, 1.0\}$) on reasoning accuracy. As shown in Table 3, there is a strict inverse correlation between temperature and accuracy.

Lowering the temperature from 0.3 to 0.1 yielded further marginal gains for the best-performing models (e.g., SGD Epoch 2 improved from 46.00% to 46.40%). This confirms that mathematical reasoning tasks benefit from minimizing stochasticity to maintain the integrity of the logical chain.

Table 3. Impact of Temperature on Accuracy (SGD Models)

Model	Temp=0.1	Temp=0.3	Temp=0.7	Temp=1.0
SGD (Ep 2)	**46.40%**	46.00%	37.20%*	25.00%
SGD (Ep 3)	44.00%	**46.20%**	33.00%	23.60%

*Note: Max tokens set to 1024 for T=0.7 comparison.

3.4. Impact of Max Generation Length

Mathematical problems often require long chains of thought. We compared the performance at $T = 0.7$ with maximum new tokens limit set to 512 versus 1024. Increasing the token limit from 512 to 1024 consistently improved performance. For instance, the SGD (Epoch 2) model improved from **31.40%** to **37.20%**. This indicates that truncation at 512 tokens prematurely cuts off the reasoning process for complex problems, resulting in incorrect or missing answers.

3.5. Resource Efficiency Analysis

While increasing the generation length improves accuracy, it comes with significant computational costs. Table 4 compares the inference time and memory usage for a batch size of 32.

Table 4. Inference Resource Usage (Batch Size=32)

Max Tokens	Avg. Runtime	Peak Memory (MiB)
512	03 min 38 s	8053
1024	09 min 05 s	17,205

Generating 1024 tokens takes approximately **2.5**× longer than 512 tokens and consumes significantly more VRAM (reaching 17GB). Therefore, while long-context generation is beneficial for accuracy, it requires careful resource management in deployment environments.

3.6. Case Study: Qualitative Analysis

To provide deeper insights into the model's reasoning capabilities, we conducted a qualitative analysis of the generated outputs. Table 5 presents a representative success case from the best-performing model (SGD, Epoch 3).

Analysis: As shown in Table 5, the finetuned model demonstrates robust chain-of-thought capabilities.

- **Methodological Generalization:** Interestingly, the model adopted a different approach than the ground truth. While the ground truth used a decomposition method (calculating distance for 2 hours and 3/4 hours separately), the model utilized the standard algebraic formula ($d = vt$) by converting the mixed number to an improper fraction ($\frac{11}{4}$). This indicates that the model has internalized the general mathematical principles rather than memorizing specific solution patterns.
- **Format Compliance:** The model strictly followed the instruction to "reason step by step" and successfully encapsulated the final result in the `\boxed{}` format, which is crucial for the automated evaluation system.
- **Arithmetic Precision:** The intermediate steps ($\frac{550}{4} \rightarrow \frac{275}{2} \rightarrow 137\frac{1}{2}$) were executed without calculation errors, showcasing the effectiveness of the SGD optimizer in preserving the model's numeracy skills during finetuning.

4. Conclusion

Experimental results demonstrate that the momentum-based SGD optimizer exhibited the best training stability and final performance, achieving the highest accuracy of 46.40%, which significantly outperforms AdamW (31.20%) and LoRA (24.00%). SGD showed a smooth convergence trend in validation loss without signs of overfitting, proving that its robustness is superior to adaptive methods and parameter-efficient fine-tuning when performing full-parameter fine-tuning on such small-scale models for mathematical tasks.

Regarding resource efficiency, LoRA technology successfully reduced VRAM requirements to approximately 5.6 GB (a reduction of about 38% compared to full fine-tuning), making training on consumer-grade GPUs possible. However, the study found that LoRA did not strictly reduce training time; instead, it slightly increased duration due to additional computational overhead, and its restricted parameter update space resulted in performance inferior to full fine-tuning. This indicates that full-parameter fine-tuning remains the pre-

ferred choice when pursuing maximum reasoning accuracy, while LoRA serves as a compromise for scenarios with extremely limited hardware resources.

Furthermore, the analysis of the inference phase emphasized the importance of determinism and long context in mathematical reasoning. Experiments confirmed that a low temperature setting ($T = 0.1$) minimizes stochasticity and enhances logical consistency, while extending the maximum generation length to 1024 tokens—despite increasing inference costs—effectively avoids the premature truncation of complex reasoning processes and significantly improves accuracy. Qualitative analysis further confirms that the fine-tuned model has internalized general algebraic problem-solving principles rather than merely memorizing patterns, providing important empirical references for optimizing mathematical reasoning models in computationally constrained environments.

Problem ID 68:

At 50 miles per hour, how far would a car travel in $2\frac{3}{4}$ hours? Express your answer as a mixed number.

Model Prediction (SGD, Epoch 3) - Correct

To find the distance traveled by the car, we can use the formula:

$$\text{Distance} = \text{Speed} \times \text{Time}$$

Given: - Speed = 50 miles per hour - Time = $2\frac{3}{4}$ hours First, convert the mixed number $2\frac{3}{4}$ to an improper fraction:

$$2\frac{3}{4} = \frac{11}{4} \text{ hours}$$

Now, multiply the speed by the time:

$$\text{Distance} = 50 \times \frac{11}{4}$$

To multiply these, we can write 50 as a fraction:

$$50 = \frac{50}{1}$$

So,

$$\text{Distance} = \frac{50}{1} \times \frac{11}{4} = \frac{50 \times 11}{1 \times 4} = \frac{550}{4}$$

Next, simplify $\frac{550}{4}$:

$$\frac{550}{4} = \frac{275}{2}$$

Finally, convert $\frac{275}{2}$ to a mixed number:

$$\frac{275}{2} = 137\frac{1}{2}$$

Therefore, the distance traveled by the car is $\boxed{137\frac{1}{2}}$ miles.

Ground Truth Reference:

In two hours, a car traveling at 50 mph would travel $50 \times 2 = 100$ miles. Now we find how far a car can travel in $3/4$ of an hour which is $50 \times \frac{3}{4} = \frac{150}{4} = 37\frac{1}{2}$ miles. Thus, the car would travel a total of $100 + 37\frac{1}{2} = \boxed{137\frac{1}{2}}$ miles.

Table 5. A successful reasoning example from the SGD fine-tuned model. The model correctly identifies the physics formula, performs fraction conversion, and arrives at the correct mixed number format.