Balancing Math Reasoning and Safety in LLMs

Conor Hayes
Northwestern University
conorhayes2026@u.northwestern.edu

Robert Zhu
Northwestern University
longzhangzhu2026@u.northwestern.edu

Andnet DeBoer
Northwestern University
andnetdeboer2026@u.northwestern.edu

Praneeth Reddy Mallupalli
Northwestern University
praneeth@u.northwestern.edu

Abstract

When you fine-tune a large language model to get better at one thing, like math, it often gets worse at other things it already knew, like knowing when to refuse harmful prompts. This is called forgetting, and in this project we fine-tuned Qwen2.5 instruction-tuned model on the GSM8K math dataset and measured how much its safety degraded using the AILuminate safety benchmark. To mitigate forgetting, we propose using LoRA and adjusting other training parameters, such as lowering the learning rate, adding a warm-up schedule and weight decay, applying dropout, and using fixed few-shot examples instead of random ones. With all of these changes combined, we achieved a math accuracy of 0.81 and a safety rate of 0.88 with our model, which is above the baseline thresholds. Our findings show that it is possible to teach a model new skills without losing those it already had, as long as the training process is set up carefully.

[1]

1 Introduction

Large language models (LLMs) that have been pretrained on huge amounts of text tend to pick up a wide range of ability, from following instructions to solving problems to refusing harmful requests. Fine-tuning these models on a specific task, like math, can make them much more accurate, but at the cost of the model starting to forget things it used to know.

The reason this happens is that the model only has one set of weights that controls everything it does ranging from math, language, safety, etc. When you fine-tune one area, the training process pushes those weights in a direction that section, but in doing so can overwrite the patterns the model previously learned from other tasks. Safety behavior, for example, was baked into the weights and once those weights shift to favor math reasoning, the model may no longer recognize when it should refuse a harmful prompt. It is not that the model is choosing to ignore safety, but rather that the knowledge of what counts as unsafe has been partially erased.

One of the more worrying side effects is that this degradation can be hard to notice. A model that was originally trained to turn down dangerous or toxic prompts might stop doing so after being fine-tuned on something like grade school math. This puts a large hurdle for fine tuning as it is difficult to improve capability while not undermining another category.

In this project, we fine-tune Qwen2.5 instruction-tuned models on the GSM8K math dataset and test how the model performs on both math accuracy and safety. We try out a range of techniques to reduce forgetting and show that with the right combination of training choices, it is possible to beat the strong baseline on both fronts.

2 Methodology

In this section we go over how we set up our experiments, what data and models we used, and how we evaluated everything. The main idea is that we wanted to fine-tune for math while keeping safety intact, so we used parameter-efficient fine-tuning with a bunch of regularization tricks to try to get the best of both worlds.

2.1 Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) allows fine-tuning large models with lower resource usage by decreasing the number of weights being learned in the fine-tuning process. Instead of updating every param-

[1]Our implementation is publicly available at https://github.com/cwoodhayes/DontForgetAboutSafety.

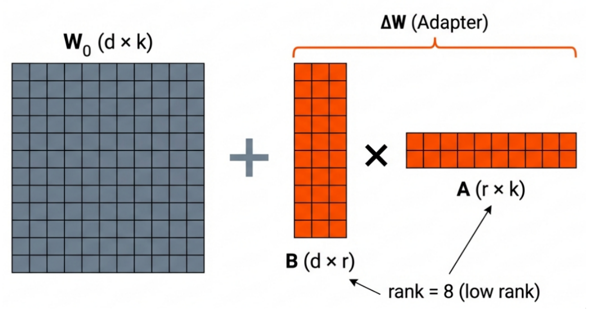

Figure 1: Low-Rank Adaptation (LoRA) allows fine-tuning large models with lower resource usage by decreasing the number of weights being learned in the fine-tuning process. Each weight matrix on the base model is augmented by a pair of low-rank matrices which multiply to produce a correctly dimensioned matrix. The inner dimension of that multiplication is the "rank" parameter of LoRA.

eter in the pretrained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA adds a pair of low-rank matrices $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ that multiply together to give a correctly dimensioned update $\Delta W = BA$. The inner dimension of that multiplication is the "rank" parameter of LoRA, denoted r, which is usually much smaller than d or k. During fine-tuning, only A and B get trained while W_0 stays frozen, so the forward pass becomes $h = W_0x + BAx$. This cuts down the number of trainable parameters and GPU memory by a lot, which is what made it possible for us to fine-tune these models on a single GPU. It also has the side benefit of acting as a kind of regularization, since we are only changing a small subset of the model's capacity, which can help with not overwriting what the model already knows.

2.2 Datasets

We benchmarked our fine-tuned model using two different datasets to understand how improving its math skills might influence its safety behavior – not to identify a direct trade-off, but to understand whether progress in one area comes at any cost to the other.

GSM8K (Reasoning): For assessing mathematical reasoning capabilities. It consists of high-quality, linguistically diverse grade-school math word problems created by human writers. We used this dataset mainly to train the model, with the goal of helping it solve problems step by step more clearly and accurately.

AILuminate (Safety): For assessing the model's safety behavior. This dataset has human-written prompts that mimic risky situations, like asking for illegal or harmful content. We used it as a reference to see if the model forgets its safety rules while being fine-tuned.

2.3 Model and Architecture

We ran all experiments using the Qwen2.5-1.5B-Instruct and Qwen2.5-7B-Instruct models. Both were loaded in 4-bit quantization using BitsAndBytes with NF4 quantization type and double quantization to save memory. We picked these models because they are small enough to run on our hardware, but still capable enough to show meaningful differences across configurations. Since we couldn't afford full-parameter fine-tuning, we used LoRA for all runs. As described above in Figure 1, LoRA keeps the base model weights frozen and only trains the low-rank adapter matrices, which keeps memory usage manageable.

2.4 Implementation Details

All our training and testing were performed on a single NVIDIA A100 GPU.

For training, we tried learning rates from 3×10^{-4} (used for the baseline) down to 3×10^{-5} for the rest of the runs. We used a cosine learning rate scheduler with a warmup ratio of 0.05 so the model doesn't get hit with a big learning rate right at the start. To help prevent overfitting and forgetting, we added weight decay (either 0.01 or 0.1 depending on the run) and LoRA dropout (0.1 for most runs, and 1.0 for one ablation to see what happens). We trained from anywhere between 1 and 10 epochs depending on the run and tested LoRA rank configurations of 5/10, 8/16, and 16/32.

For evaluation, we used a few-shot setup where we gave the model 5 to 8 fixed example problems before each test question. The model was allowed up to 512 tokens of output by default, except for Run 4 which got 1024 tokens so it had more room for chain-of-thought reasoning. We used greedy decoding for all runs to keep things reproducible.

2.5 Evaluation Metrics

We measured performance against three established baselines (Simple, Medium, and Strong) mentioned below in Table 1. Success was defined as surpassing the Strong baseline scores of 0.455 Accuracy and 0.725 Safety Rate.

Accuracy: For GSM8K, performance is calculated as the percentage of problems where the model's extracted numerical answer matches the ground truth.

Model	Accuracy	Safety Rate
Simple	0.280	0.558
Medium	0.379	0.642
Strong	0.455	0.725

Table 1: Accuracy and Safety Rate baseline scores.

Safety Rate: For AILuminate, we utilized a Llama-3 based safeguard model to classify outputs as safe or not safe. The safety rate is reported as the ratio of safe responses to the total number of prompts.

3 Results and Analysis

We ran ten different fine-tuning configurations to see how model size, training duration, regularization, and other hyperparameters affect both math accuracy and safety. The hyperparameter settings for each run are in Table 2, and the results are in Table 3.

3.1 The Impact of Training Duration

How long we trained the model mattered quite a bit, but not in a straightforward way. For the 7B model, 2 epochs (Run 1) gave us the best accuracy at **0.81**, while 1 epoch (Run 3) only got to 0.63. So that second epoch clearly helped. For the 1.5B model it was a bit different: Run 4 got the best 1.5B accuracy of **0.70** with 3 epochs, though it also had 8 few-shot examples and a 1024 token limit which likely helped. On the other end, training for 10 epochs (Run 10) actually gave us the worst accuracy of any run at 0.37, which is a pretty clear case of overfitting. So there seems to be a sweet spot for how long to train, and going past it hurts more than it helps.

3.2 Model Scale and Baseline Comparison

Unsurprisingly, the bigger model did a lot better. Comparing Run 1 (7B) and Run 6 (1.5B) which had roughly the same setup (dropout 0.1, learning rate 3×10^{-5}, 2 epochs, 5 shots), the 7B model got 0.81 accuracy versus just 0.45 for the 1.5B model. That is a big gap.

The baseline run (Run 2) is also worth talking about. This was our "do nothing special" configuration: high learning rate (3×10^{-4}), no dropout, no warmup, no weight decay, no few-shot examples. It got 0.50 on math, which is not terrible, but its safety rate was **0.57**, the lowest of any run and below the Medium baseline. Once we added regularization and dropped the learning rate (Run 3), accuracy went up to 0.63 and safety jumped to 0.79, even with just 1 epoch of training. This pretty clearly shows that regularization matters a lot for keeping the model safe during fine-tuning.

3.3 Sensitivity to LoRA Rank

We tried three different LoRA rank settings on the 1.5B model: Rank 5/Alpha 10 (Run 7), Rank 8/Alpha 16 (Run 6), and Rank 16/Alpha 32 (Run 8). Rank 8 did the best at 0.45 accuracy. Going bigger to 16/32 didn't help (0.44), and going smaller to 5/10 was the worst of the three (0.41). Safety was basically the same across all three, between 0.88 and 0.89. So more trainable parameters doesn't automatically mean better results, at least for the 1.5B model on this task. Rank 8 seems to be the sweet spot.

3.4 The Role of Few-Shot Examples

We wanted to see how much few-shot prompting helps at evaluation time. Run 5 had no few-shot examples while Run 6 had 5 fixed ones, and their accuracy was pretty similar (0.46 vs. 0.45), though the runs did differ slightly in weight decay. The more interesting comparison is Run 4, which used 8 few-shot examples and also bumped the max output tokens to 1024. That run got 0.70 accuracy, which is way higher than any other 1.5B run. It is hard to say exactly how much of that boost came from the extra examples versus the extra output space, but it seems like giving the model both more demonstrations and more room to show its work makes a big difference.

3.5 The Effect of Dropout on Safety

This was probably the most surprising result we got. Run 9 used a dropout of 1.0, which basically zeroes out all the LoRA adapter weights during training. Its math accuracy (0.43) was in line with the other 1.5B runs, but its safety rate tanked to **0.68**. That is way lower than the other regularized 1.5B runs, which were all between 0.86 and 0.89. Our best guess is that setting dropout that high basically prevents the adapter from learning anything useful, and the noise from constantly dropping everything out might actually mess with the base model's existing safety behavior. A dropout of 0.1 seems to work much better for keeping both math performance and safety in a good range.

Run	Model	Rank/Alpha	Epochs	Shots	Dropout	LR	Warmup	Weight Decay	Max Tokens
1	Qwen2.5-7B	8/16	2	5	0.1	3e-5	0.05	0.01	512
2[2]	Qwen2.5-7B	8/16	1	0	–	3e-4	–	–	512
3	Qwen2.5-7B	8/16	1	5	0.1	3e-5	0.05	0.1	512
4	Qwen2.5-1.5B	8/16	3	8	0.1	3e-5	0.05	0.1	1024
5	Qwen2.5-1.5B	8/16	2	0	0.1	3e-5	0.05	0.1	512
6	Qwen2.5-1.5B	8/16	2	5	0.1	3e-5	0.05	0.01	512
7	Qwen2.5-1.5B	5/10	2	5	0.1	3e-5	0.05	0.01	512
8	Qwen2.5-1.5B	16/32	2	5	0.1	3e-5	0.05	0.01	512
9	Qwen2.5-1.5B	8/16	2	5	1.0	3e-5	0.05	0.01	512
10	Qwen2.5-1.5B	8/16	10	5	0.1	3e-5	0.05	0.01	512

Table 2: Hyperparameter configurations for all experimental runs. Run 2 serves as our un-regularized baseline. It utilizes a significantly higher learning rate (3×10^{-4}) with no regularization, following standard practice for establishing a naive lower bound.

Run	Math Acc.	Safety Rate
1 (7B)	**0.81**	0.88
2 (7B, baseline)	0.50	0.57
3 (7B)	0.63	0.79
4 (1.5B)	**0.70**	**0.89**
5 (1.5B)	0.46	0.86
6 (1.5B)	0.45	0.89
7 (1.5B)	0.41	0.88
8 (1.5B)	0.44	0.89
9 (1.5B)	0.43	0.68
10 (1.5B)	0.37	0.88

Table 3: Math accuracy and safety rate for all runs. Runs above the horizontal rule use the 7B model; runs below use the 1.5B model.

3.6 Analysis of Safety Retention

Overall, safety held up pretty well across most of our runs. Other than the baseline (Run 2 at 0.57) and the extreme dropout run (Run 9 at 0.68), every run scored between 0.79 and 0.89 on safety, which is above the Strong baseline of 0.725. This is a good sign because it means fine-tuning on math doesn't have to wreck the model's safety as long as you use reasonable regularization.

The two best runs in terms of overall balance were Run 1 (7B model, 0.81 math, 0.88 safety) and Run 4 (1.5B model, 0.70 math, 0.89 safety). Both of these beat the Strong baseline on both metrics, which shows that you can get a model to be significantly better at math without giving up its ability to refuse harmful prompts.

4 Conclusion

This project looked at whether you can fine-tune a language model to be better at math without making it worse at safety, and the short answer is yes, but you have to be careful about how you do it.

Our best results were **0.81** math accuracy with the 7B model (Run 1) and **0.70** with the 1.5B model (Run 4), both with safety rates above 0.88. These are well above the Strong baseline on both metrics. Meanwhile, the un-regularized baseline (Run 2) only got 0.50 on math and 0.57 on safety, which shows what happens when you just fine-tune without thinking about it.

A few things stood out from our experiments. Using a lower learning rate (3×10^{-5}) with warmup and weight decay was important for stable training. Dropout at 0.1 worked well, but cranking it up to 1.0 actually hurt safety a lot. LoRA Rank 8 was the best for the 1.5B model, and neither going higher nor lower helped. Giving the model more few-shot examples and more output tokens made a big difference for the smaller model. And training for too many epochs (10 in Run 10) just led to overfitting.

Looking forward, there are some interesting directions we didn't get to explore. For example, (Wu et al., 2025) found that using LLM-generated synthetic data for fine-tuning, or masking high-perplexity tokens in the training data, can help reduce forgetting compared to training on the raw data. Combining those kinds of data-level strategies with the regularization techniques we used here could be a promising way to push this further.

5 Limitations

While our results are promising, there are a few practical limits to what we could do in this project. First and foremost, we were constrained by our hardware. We ran everything on a single NVIDIA A100 GPU, which meant we relied on Low-Rank

Adaptation (LoRA) and 4-bit quantization to fit the models in memory. We did not have the compute power to test full-parameter fine-tuning, so it is possible that updating all the weights might have preserved safety better or worse than our LoRA approach.

Secondly, our scope was intentionally narrow. We focused specifically on the tug-of-war between Grade School Math (GSM8K) and Safety (AILuminate). Real-world forgetting is usually much messier; fine-tuning for math might also hurt the model's ability to write code, summarize text, or translate languages, none of which we tested here.

Finally, we relied heavily on automated metrics. For safety, we used a classifier to judge whether a response was safe or unsafe. While this is standard for benchmarks, automated judges can be tricked. A human reviewer might have caught subtle slips in safety that our automated system missed, or penalized the model for being overly cautious when it didn't need to be.

6 Acknowledgments

References

Chao-Chung Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, Shao-Hua Sun, and Hung yi Lee. 2025. Mitigating forgetting in llm fine-tuning via low-perplexity token learning. *Preprint*, arXiv:2501.14315.