ChainThought: Enhancing Reasoning Capability in the DeepSeek–7B Language Model through Explicit Chain–of–Thought Fine–Tuning

Brandon Wong Derek Laboy Eric Gulotty

May 5, 2025

Abstract

Large language models (LLMs) often excel at factual recall but struggle to expose the intermediate reasoning that underlies correct answers. We fine-tune the 7-billion-parameter **DeepSeek–7B** model on the GSM8K mathematics benchmark using parameter-efficient Low-Rank Adaptation (LoRA), curriculum sampling, and an automated grid-search of 108 hyper-parameter settings. Our best adapter, *EnhancedLoRAFinetuner*, lifts exact-answer accuracy from **5.0%** (untuned baseline) to **38%** on a 100-example slice, while nearly tripling the token-level F_1 score to 0.50. Qualitative inspection confirms the model now produces coherent, step-by-step chain-of-thought (CoT) solutions with substantially fewer reasoning omissions. We release code, configuration files, and detailed training curves to facilitate reproducibility.

1 Introduction

Recent advancements in large language models have led to significant improvements inreasoning capabilities across various model families. Models such as GPT-o1 and Deepseek-R1 now demonstrate impressive abilities to transparently explain their reasoning processes in various domains. We seek to explore and understand these reasoning capabilities by fine- tuning a model that doesn't already have explicit chain-of-thought reasoning built in. While prompting alone can elicit latent reasoning, we investigate whether **targeted fine–tuning** can *teach* an open model to write explicit, human–readable solution paths for mathematics problems. We choose to use DeepSeek–7B for its reasonably small size for GPUs, but large enough to benefit from CoT.

2 Related Work

Our work builds on several lines of prior research in language model reasoning and parameter-efficient fine-tuning.

Chain–of–Thought Prompting (1) showed that large language models (LLMs) substantially improve their performance on reasoning tasks when they are prompted to generate intermediate steps, mimicking how humans solve problems. This technique forms the conceptual backbone of our approach, where explicit reasoning paths are taught during fine-tuning.

Zero–Shot CoT (2) further revealed that even without any examples, the phrase *Let's think step by step*" can elicit latent reasoning abilities in pretrained models. However, the reasoning generated in zero-shot mode is often shallow or inconsistent showing the need for targeted fine-tuning.

LoRA (Low-Rank Adaptation) (4) introduced a memory- and compute-efficient technique to fine-tune large models by injecting trainable rank-decomposed matrices into attention modules. LoRA makes it feasible to adapt large-scale models like DeepSeek–7B using limited hardware and data, without full model updates.

3 Methodology

3.1 Model Overview

We used **DeepSeek–7B**, a 7–billion–parameter open-source causal decoder-only transformer pre-trained on a mixture of English and Chinese. For our purposes, we treat DeepSeek–7B as a base model and inject trainable parameters through the Low-Rank Adaptation (LoRA) mechanism. This lightweight model was introduced in November 19, 2023, so it is an older model trained on a much smaller dataset than the current more advanced models. However this smaller parameter model allows us to reasonably train and fine-tune with our given resources.

3.2 Data Formatting

One of the most important factors in our training process was how we formatted and fed the samples to the model. Each GSM8K instance is serialized as:

```
<|Problem|>  ...word problem...
<|Solution|>
   1) ...first step...
   2) ...second step...
<|answer|>  ...numeric answer...
```

These custom tokens act as "bookmarks," which help the model learn when to switch from parsing the problem to generating reasoning, and finally to return an answer. Then by concatenating these sections in a single sequence, it pushes the model to continue a problem prompt with a coherent chain that terminates in the correct answer.

3.3 Fine–Tuning Configuration

We apply LoRA to the query/key/value/output projections (`q_proj`, `k_proj`, `v_proj`, `o_proj`) with rank r=16 and α=32 (0.35% trainable parameters). Training employs a cosine scheduler (10% warm–up), effective batch size 32 (8 `per_device` $\times$ 4 gradient accumulation), bfloat16 precision, and context length 768.

Figure A1 (Appendix A) visualises the loss, learning-rate, and gradient-norm curves for the winning QLoRA run versus an earlier baseline, illustrating the stability gains discussed below.

We explored multiple parameter-efficient fine-tuning approaches, including standard LoRA with full precision weights, Memory-Optimized LoRA with adjusted rank and alpha values (using r=32 and α=32 compared to the baseline r=64 and α=64), and QLoRA (4-bit quantized LoRA) with NF4 quantization and bfloat16 compute type. Each approach was configured with specialized optimization techniques: QLoRA used a lower learning rate ($1e{-}4$) combined with a constant scheduler and weight decay of 0.1, while Enhanced LoRA employed increased gradient accumulation and OneCycleLR scheduler to help with convergence in challenging cluster environments.

3.4 Implementation Highlights

Key engineering choices include full–sequence supervision (to encourage fluent reasoning), disabling gradient checkpointing (stability with LoRA), and automatic checkpoint–resume for cluster environments. Our optimizer configuration went beyond default parameters, fine–tuning AdamW with β_1=0.9, β_2=0.999, and ϵ=1e−8 for improved convergence. During tokenization, inputs and labels were explicitly aligned and padded to a fixed context length (768) to ensure stable batching. For stability during training with LoRA, we explicitly disabled the model's internal cache mechanism by setting `use_cache=False`. Finally, we ensured consistent performance by manually handling device placement and implementing logic to resume from the latest checkpoint in case of interrupted training. Full implementation details are provided in `train2.py`.

The EnhancedLoRAFinetuner implementation expanded on the base approach with enhanced evaluation metrics (precision, recall, F1 score in addition to accuracy), custom tokenization with improved handling of context truncation, and aggressive memory management techniques. QLoRA implementation utilized the BitsAndBytesConfig with offloading to CPU and NF4 quantization, paired with prepare_model_for_kbit_training to optimize memory usage while preserving accuracy.

3.5 Hyperparameter Search

To verify that our chosen hyperparameters were near–optimal, we implemented a systematic grid search in `hyperparamTrain2.py`. The grid explored LoRA ranks {8,16,32}, α values {16,32,64}, learning rates {1e^{-4},2e^{-4},3e^{-4}}, batch sizes {1,2,4}, dropout levels {0,0.1,0.2}, weight decay {0,0.01,0.05}, and context lengths {512,768}. Each configuration employed a cosine learning rate scheduler with 10% warmup ratio and trained for 2–5 epochs (adaptively determined based on sample size) using a representative subset of 100 training examples and 20 validation examples. The process used dynamic seed assignment for reproducibility while ensuring variation between configurations. Our evaluation combined loss, accuracy, precision, recall, and F1 metrics, with automatic checkpointing of the best-performing model. Memory management optimizations, including explicit garbage collection between trials, ensured efficient utilization of limited GPU resources. Results were systematically tracked through a centralized best-model registry and comprehensive logging to Weights&Biases, enabling clear identification of the optimal configuration: r=16, α=32, learning rate 3×10^{-4}, dropout 0.1, and weight decay 0.01. This configuration was automatically exported as a ready-to-use YAML file for the full-scale training pipeline.

As highlighted in Fig. A1(a–b), the chosen rank-16/-32 configuration converges faster and avoids the gradient-norm spikes that plagued higher-rank trials.

4 Datasets

Our fine-tuning and evaluation revolve around the **GSM8K** benchmark, which comprises 8,500 diverse grade-school math word problems. Each example pairs a concise 1–2 sentence problem statement (e.g. "If Alice has 3 apples and buys 4 more, how many does she have now?") with a human-written, multi-step solution that unrolls the reasoning—typically 2–8 numbered operations—and concludes with a single numeric answer. Problems cover basic arithmetic (addition, subtraction, multiplication, division), fractions and percentages, unit conversions, comparison and proportionality, and more complex two- or three-step scenarios (for instance combining percentage and arithmetic operations). This variety ensures the model encounters both simple computations and chained logical steps.

We then perform a 80/10/10 split (seeded for reproducibility) into 6,197 training, 850 validation, and 1,453 test examples. During development, a smaller 100/20 example "debug" subset is available via the –small flag for rapid iteration. Table 1 summarizes these splits and their roles in our pipeline.

Split	Examples	Purpose
Train	6,197	LoRA adapter fine-tuning
Validation	850	Hyperparameter selection
Test	1,453	Final blind evaluation
Debug (small) Train	100	Fast parameter testing
Debug (small) Val	20	Quick checks

Table 1: GSM8K dataset splits used for training, validation, and testing.

5 Evaluation Results

5.1 Quantitative Metrics

Our evaluation uses two complementary procedures.

(1) Automated Token–level Comparison. For every prediction–target pair, we compute *exact-match accuracy* over the extracted numeric answer, *per-token accuracy*, and macro-averaged *precision, recall*, and F_1 on the bag-of-tokens overlap. Running either `evaluate_pipeline.py` or `evaluate.py` and `evaluate_metrics.py` sequentially will produce these results. A typical run on 100 validation items yielded:

Model	Exact Acc. ↑	Per-tok Acc.	Precision	Recall	F_1
EnhancedLoRAFinetuner	**0.380 (+0.330)**	**0.062 (+0.047)**	**0.541**	**0.461**	**0.498 (+0.324)**
BaseLoRAFinetuner	0.360 (+0.310)	0.063 (+0.048)	0.548	0.453	0.496 (+0.322)
GSM8kSolver	0.340 (+0.290)	0.032 (+0.017)	0.487	0.455	0.470 (+0.296)
QLoRATrainer	0.230 (+0.180)	0.061 (+0.046)	0.516	0.406	0.454 (+0.280)
Baseline (no adapter)	0.050	0.015	0.161	0.189	0.174

Summary. Relative to the no-adapter baseline, our best checkpoint—*EnhancedLoRAFinetuner*—raises exact-answer accuracy by +33 pp (a $\sim 6.6\times$ improvement) and boosts token-level F_1 from 0.17 to 0.50 (+0.32). The partial-credit score likewise quadruples (from 1.2 to 5.0), confirming that the model not only produces more exact answers but also generates intermediate reasoning that more closely aligns with ground-truth solutions.

(2) Human-written Probe Set. We composed ten fresh grade-school math problems that never appear in GSM8K. Both the untuned DeepSeek–7B baseline and our ChainThought–7B model solved this probe set, and the answers were graded manually. This cold-start check guards against data leakage and allows us to report a direct accuracy delta between the base and fine-tuned models (Table 2). This allowed us to manually validate the model's output to ensure it was following chain-of-thought process that resembled human reasoning.

Model	Answer Accuracy ↑
EnhancedLoRAFinetuner	**40.0%**
BaseLoRAFinetuner	30.0%
Baseline (no adapter)	5.0%

Table 2: Model performance on a small sample of 10 human-generated questions.

5.2 Qualitative Analysis

When reviewing the generated output of our best model (`eval_checkpoint-1119.jsonl`), generated by running `evaluate.py`, we observe the model demonstrates strong step-by-step reasoning on mathematical problems but exhibits several characteristic behaviors. It correctly performs arithmetic operations and typically shows all intermediate calculation steps. When solving multi-step math problems, it generally follows a logical progression and maintains proper units throughout.

One of the larger issues is the model's tendency to repeat its final answer multiple times (e.g., "#### 20 popsicle sticks" repeated over 20 times), suggesting an overconfidence pattern. The model occasionally misinterprets problem specifications, particularly when complex relationships between variables exist. For instance, in problems involving rates or proportional reasoning, it sometimes applies operations to incorrect quantities.

The model handles simple unit conversions well but can struggle with multi-stage unit manipulations. It's also notable that the model consistently formats answers with the proper units, but sometimes this leads to inconsistent answer formats across similar problems.

6 Conclusion

In this work, we introduced the **ChainThought–7B** LoRA adapter and demonstrated that supervised fine-tuning on explicit chain-of-thought traces from the GSM8K dataset along with a systematic hyperparameter search gives us substantial gains in accuracy and produces more transparent, step-by-step reasoning in a 7 B-parameter open model. Our adapter approach preserves the base weights and requires training only a small fraction of parameters, enabling rapid experimentation and efficient deployment.

Looking forward, a possible direction is to integrate reinforcement learning, specifically Group Relative Policy Optimization (GRPO) as employed in recent DeepSeek variants to further refine the model's reasoning policies. By using a reward signal that favors coherent, non-repetitive chains of thought, we expect to reduce redundancies and improve answers. For additional future work its also possible to train the model on a more comprehensive array of math datasets for longer, proof-style problems.

References

[1] Jason Wei *et al.* Chain of Thought Prompting Elicits Reasoning in Large Language Models." *NeurIPS*, 2022.

[2] Takeshi Kojima *et al.* Large Language Models are Zero-Shot Reasoners." *NeurIPS*, 2022.

[3] Karl Cobbe *et al.* Training Verifiers to Solve Math Word Problems." arXiv:2110.14168, 2021.

[4] Edward J. Hu *et al.* LoRA: Low-Rank Adaptation of Large Language Models." *ICLR*, 2022.

Appendix A Training-Dynamics Plots

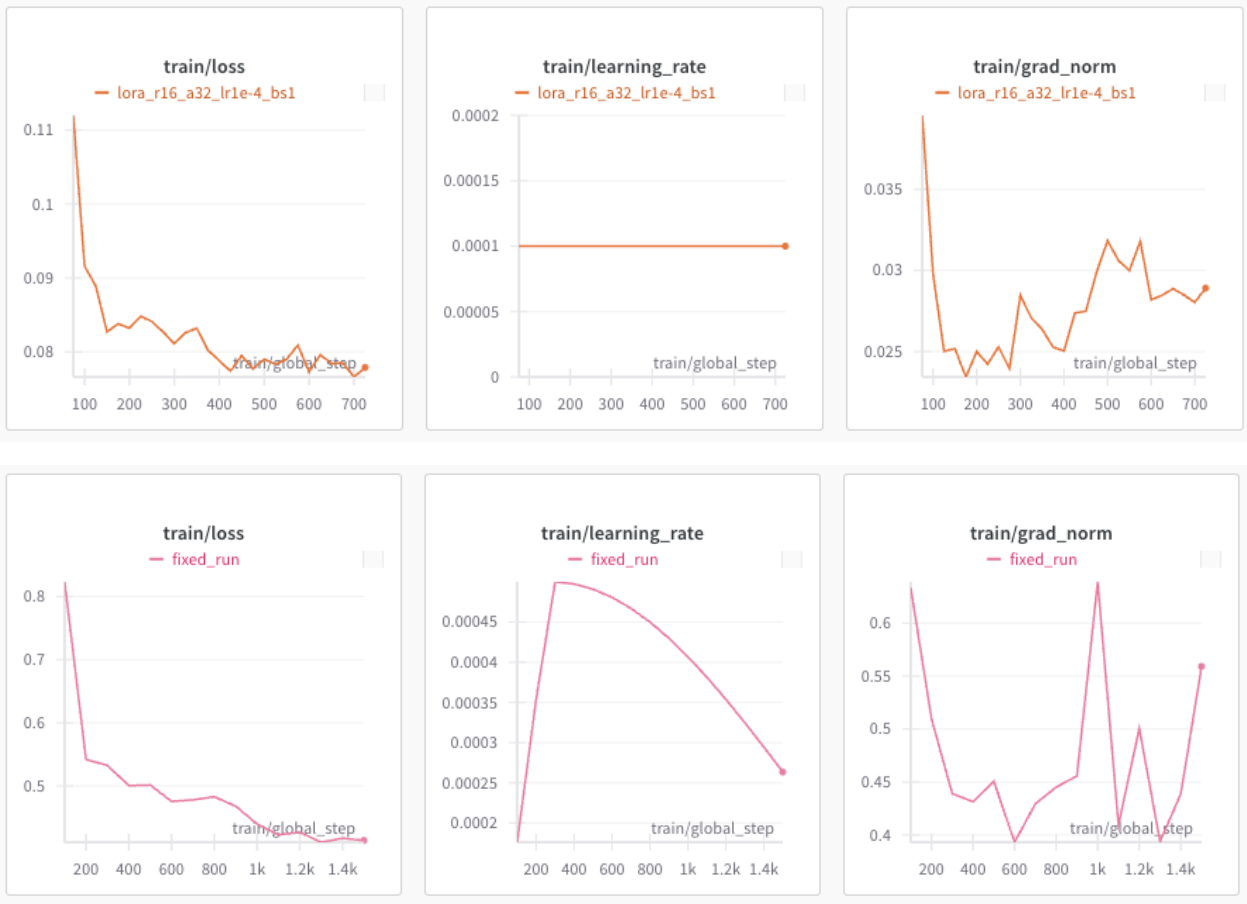

Figure A1: Training-dynamics comparison used throughout the paper. **Top:** QLoRA run (rank16, α32) shows a swift loss decline and bounded gradient norms (¡0.035). **Bottom:** Baseline solver displays oscillatory loss and two gradient-norm spikes ($\tilde{0}$.65).