An Empirical Study of RLVR Fine-Tuning for Mathematical Problem Solving in LLMs

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

Master of Technology

in

Computer Science

submitted by

Rashmi Konnur

(Roll No. CS2423)

under the supervision of:

Prof. Utpal Garain

Computer Vision & Pattern Recognition Unit,

Indian Statistical Institute, Kolkata

Indian Statistical Institute

Kolkata-700108, India

June 2026

Declaration of Authorship

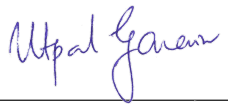
I, **Rashmi Konnur**, declare that this Master's thesis titled, *An Empirical Study of RLVR Fine-Tuning for Mathematical Problem Solving in LLMs* is a result of my own research carried out under the guidance of Prof. Utpal Garain. To the best of my knowledge, this work contains no materials previously published or written by any other person, or substantial proportions of material which has formed the basis of award of any other degree or diploma at ISI Kolkata or any other educational institution, except where due acknowledgment is made in this thesis. I authorize ISI Kolkata to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Date: 10 June 2026

Rashmi Konnur
Roll No: CS2423
Mtech in Computer Science
Indian Statistical Institute, Kolkata

Certificate of the Supervisor

This is to certify that the dissertation titled **An Empirical Study of RLVR Fine-Tuning for Mathematical Problem Solving in LLMs** submitted by **Rashmi Konnur** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by her under our supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in our opinion, has reached the standard needed for submission.



10/June/2026

Prof. Utpal Garain
Computer Vision & Pattern Recognition Unit
Indian Statistical Institute, Kolkata

Acknowledgments

I thank my advisor, *Prof. Utpal Garain*, Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, Kolkata for his invaluable guidance during the course of my progress on this thesis. His unwavering support and patience have helped in shaping this work. I thank Dr. Soumadeep Saha for introducing me to the problem.

I sincerely thank all the faculty members and researchers at the Indian Statistical Institute for cultivating an inquisitive atmosphere and providing ample resources for a rich academic environment. I would also like to thank my classmates for their camaraderie and constructive discussions. Our shared enthusiasm for learning has been a motivating force.

Rashmi Konnur
Indian Statistical Institute
Kolkata 700108, India.

Contents

Abstract

Large language models have shown immense improvement in coding and math performances thanks to reinforcement learning boosted algorithms. However, its true impact on broadening the reasoning and analytical capacities of an LLM is still contended. In this dissertation, we outline the foundations of Large Language Models, and delve into Reinforcement Learning with Verifiable Rewards (RLVR). We discuss various strategies to efficiently manipulate memory during a fine tuning update. We finally perform RLVR fine-tuning techniques on different models with varied use cases and compare their performances, which corroborate the efficiency of RLVR.

Keywords: RLVR, LLM, LoRA, QLoRA, Math Benchmarking

—

Chapter 1

Introduction

1.1 Introduction

Artificial Intelligence is technology that enables computers to mimic human intelligence and decision making.[1] Devices equipped with AI can learn and act based on new information, and subsequently replace human intervention. Some notable examples of this technology include navigational devices, recommender systems, facial recognition, etc.). Machine Learning is a subset of Artificial Intelligence where we algorithmically discern relevant patterns in a given dataset and make suitable inferences in similar, but unseen data. The origin of the term is often attributed to Arthur L. Samuel's 1959 article in IBM Journal, "Some Studies in Machine Learning Using the Game of Checkers." All machine learning can be categorized into three different learning paradigms:

- **Supervised Learning:** involves training a model against the ground truth, optimizing for the accuracy rate for prediction of the correct label. E.g., classification tasks and regression.
- **Unsupervised Learning:** involves training a model to discern the intrinsic dependencies in a dataset. There is no external ground truth against which the outputs are compared. E.g., k-means clustering, dimensionality reduction algorithms.
- **Reinforcement Learning:** instead of using a single ground truth, training the model involves evaluating the environment and and optimizing through a series of actions the greatest reward.

1.2 What is Deep Learning?

It is the subset of Machine Learning that leverages multi-layered neural network. 'Deep learning' refers to the training of models with at least 4 layers of neural networks, although modern neural networks are much 'deeper'[1]. Deep neural networks are universal approximators i.e., given any continuous function and a margin of error ϵ, there exists a neural network design which can approximate it within this bound.

1.2.1 Universal Approximation Theorem

Between the 1980s and the 1990s, a slew of papers were published that established several variants of the universal approximation theorem with restrictions on layer widths, depths and the family of activation functions. However the most classic version of the theorem states that neural networks are dense in the space of continuous functions under the supremum norm.[2] Mathematically, for a compact subset K, a continuous function f and a margin of error ϵ, there exists a neural network $\hat{f}$ such that:

$$sup_{x \in K} |f(x) - \hat{f}(x)| < \epsilon \tag{1.1}$$

Here(Cybenko, 1989), the family of neural network functions are finite sums of the form:

$$\hat{f}(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j) \tag{1.2}$$

where $\alpha_j \in \mathbb{R}, y_j^T, x, \theta_j \in \mathbb{R}^n$ and σ is a sigmoidal function (continuous) defined as:

$$\sigma(t) \to \begin{cases} 1 & \text{as } t \to +\infty, \\ 0 & \text{as } t \to -\infty. \end{cases} \tag{1.3}$$

The original rudimentary proof uses Hahn-Banach theorem and Riesz Representation theorems to prove for this specific class of activation functions. Contemporary works soon made it clear that it was not the specific choice of activation functions but the multilayer feedforward architecture itself that drives the potential[3]. Hornick(1991) generalized on previous existing works by showing that the result holds true for all non-constant, bounded and continuous activations, not just sigmoids[4]. However, the existence theorems do not lend themselves to the realization of an actual network faithful to a given ϵ. In practical applications the neural networks primarily achieve pointwise convergence at the dataset level to the target function especially as network capacity and the dataset sizes go up[5].

1.3 What is an LLM

A language model is a computational model that predicts sequences in natural language. More precisely, language modeling is the foundational task of estimating the probability distribution over words or tokens i.e., we have to find a $\hat{f}$ such that:

$$[\hat{f}(w_{T-1}, \ldots, w_1)]_{w_T} \approx P(w_T \mid w_{T-1}, w_{T-2}, \ldots, w_1) \tag{1.4}$$

Language models have an immense span of uses in the modern world, including but not limited to natural language generation, sentiment analysis, machine translation, code generation and conversational AI like chatbots.

1.3.1 N-gram models

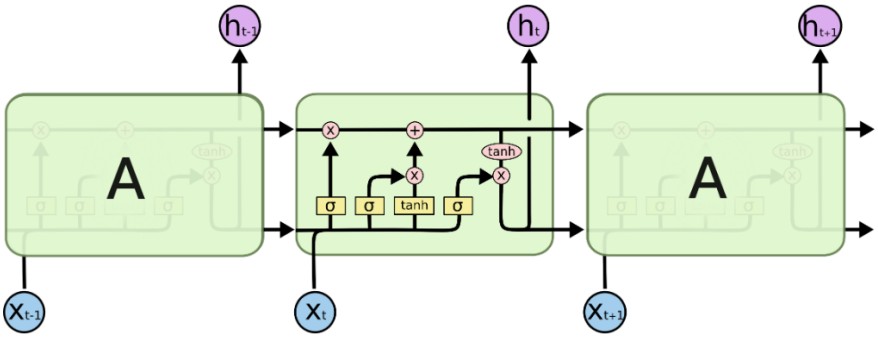
The earliest approaches to language modeling were based on the Markovian style that sought to restrict the length of the history of tokens considered. An n-gram model works on the premise that prediction of the next word depends only on the n-1 words that precede it.

$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1}) \tag{1.5}$$

Even though these preliminary models had the charm of simplicity, they display abysmal failure in capturing the long range dependencies that is pivotal in a meaningful manipulation of natural language.

1.3.2 Neural Language models

RNN (recurrent neural network) and LSTM (long short term memory) parametrized the context using a hidden state vector. LSTMs are a special kind of RNNs that are much more effective at learning long term dependencies[6].



Courtesy: https://colah.github.io

$$h_t = \text{LSTM}(x_t, h_{t-1}) \tag{1.6}$$

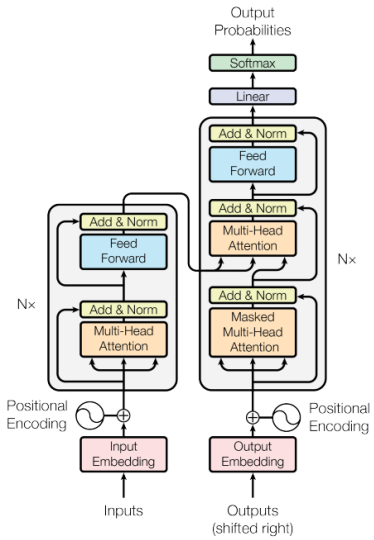
$$P(w_t \mid w_1, \ldots, w_{t-1}) = \text{softmax}(W_v h_t + b) \tag{1.7}$$

where W_v is the vocabulary projection matrix, which is a learnable weight matrix that maps the low dimensional hidden state vector h_t to a high dimensional space that represents the entire vocabulary space V.

Although this was a landmark improvement over the previous language models, what we gained in reliable historical context, we lost in mass scalability. The process relied on sequential processing of data which greatly prohibited large-scale implementations.

1.3.3 The Transformer and the advent of the Large Language Model

The introduction of self-attention[7] greatly helped overcome the sequential dependencies of the LSTM variants. This enables parallelization which in turn can help us process large amounts of data. The Scaled Dot-Product Attention maps a set of queries Q, keys K, and values V as follows:



Courtesy: Attention is all you need[7]

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.8}$$

where d_k is the scaling dimensionality of the keys. Modern LLMs parametrize these representations across billions of parameters which help in aggregating vast in-context learning capabilities.

Table 1.1: Comparison of Language Modeling Paradigms

Era	**Core Architecture**	**Context Mechanism**	**Primary Limitation**
Statistical	n-grams / Markov Chains	Fixed history window	High data sparsity
Deep Recurrent	RNN / LSTM / GRU	Sequential hidden state	Vanishing gradients
Transformer	Autoregressive Attention	Parallelized self-attention	Quadratic complexity $\mathcal{O}(T^2)$

1.4 What is Chain of Thought Prompting?

Chain-of-thought prompting is a prompting strategy that enhances the reasoning capabilities of an LLM[8]. Instead of forcing a direct output for a given input prompt query, CoT elicits a sequence of intermediate natural language reasoning steps which is followed by the final output. There isn't a consensus on the precise mechanics of CoT, however, it is theorized that the following occurs:

In standard prompting, a language model parametrizes the conditional probability of a token

sequence response Y given an input prompt X:

$$P(Y \mid X) \tag{1.9}$$

But in contrast, CoT prompting introduces a sequence of intermediate reasoning steps denoted as $C = (c_1, c_2, \ldots, c_n)$.

$$P(C, Y \mid X) = \left(\prod_{i=1}^{n} P(c_i \mid X, c_1, \ldots, c_{i-1}) \right) \cdot P(Y \mid X, C) \tag{1.10}$$

Hence, by conditioning the final prediction Y on the explicit context C, we force the model to allocate more tokens to the reasoning process, reducing any significant logical fallacies.

```
# Identity

You are a helpful AI model. Assist the user with their math queries.

# Instruction

* Solve the math problem entered by the user.
* Answer after thinking step by step.
* Provide the final answer in a box

# Example 1
<user>
Weng earns $12 an hour for babysitting.
Yesterday, she just did 50 minutes of babysitting. How much did she earn?
</user>
<assistant>
Weng earns 12/60 = $0.2 per minute.
Working 50 minutes, she earned 0.2 x 50 = $10.
Therefore the answer is $10.
$\boxed${10}$
</assistant>
```

Example: We use instruction prompting and few-shot prompting to leverage CoT in LLM

1.4.1 Self-Consistency Chain-of-Thought (CoT-SC)

As described above, standard CoT uses a linear pathway to reach the final conclusion ($X \to C \to Y$). If a single intermediate reasoning token is faulty, it can potentially derail the entailing logical steps till the conclusion. To circumvent this issue, we look at ways of treating reasoning as a multi-path exploration. Proposed by Wang et al.[9], this improvement seeks to replace the greedy decoding that is used in CoT prompting by first sampling a variety of reasoning paths $\{C^{(m)}, Y^{(m)}\}_{m=1}^{M}$ and then selecting the most consistent answer. The final output choice uses a marginalization step that sums up the probabilities of all different reasoning paths that lead to the exact same final answer, collapsing the path details to isolate the answer itself. Instead of picking the single reasoning path with the highest individual probability, it aggregates the total confidence score for each unique final answer Y across all M generated chains.

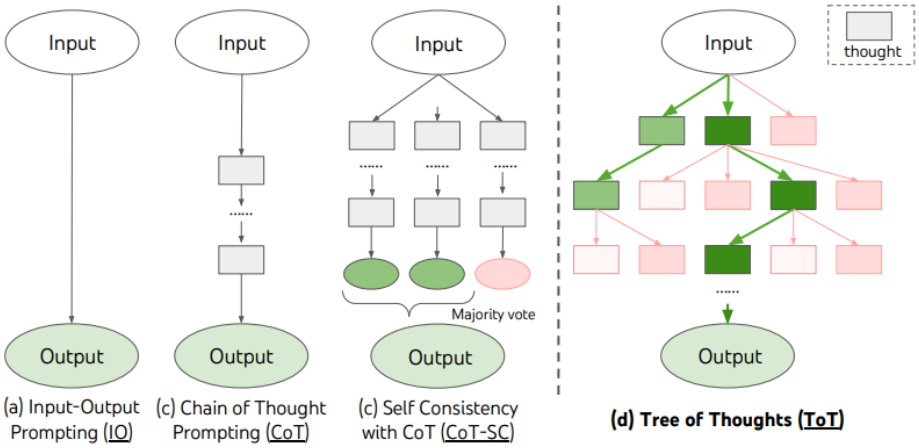
$$\hat{Y} = \arg\max_{Y} \sum_{m=1}^{M} \mathbb{I}\left(Y^{(m)} = Y\right) P\left(C^{(m)}, Y^{(m)} \mid X\right) \tag{1.11}$$

where $\mathbb{I}$ is the indicator function. This approach filters out isolated erratic paths by anchoring the output to the most consistent statistical mode.

1.4.2 Tree of Thoughts (ToT)

Yao et al. [10] generalized the CoT into a tree-like structure where instead of treating a thought sequence as an atomic unit, ToT breaks down the problem space into several discrete nodes i.e., a state $s = [x, z_{1,\dots i}]$ representing a partial solution (an input along with the thought processed till stage i). At each node the language model performs a dual role:

1. **Thought Generation:** Generating a set of potential next-step candidates given the active path, $P_\theta^{CoT}(z_{t+1} \mid X, z_1, \dots, z_t)$.
2. **State Evaluation:** Acting as a heuristic evaluator to score or vote on the viability of each node, e.g., a value prompt evaluating the state s to generate a score on the scale 1-10



Courtesy: Tree of Thoughts: Deliberate Problem Solving with Large Language Models[10]

Chapter 2

Background

2.1 Reinforcement Learning in LLMs

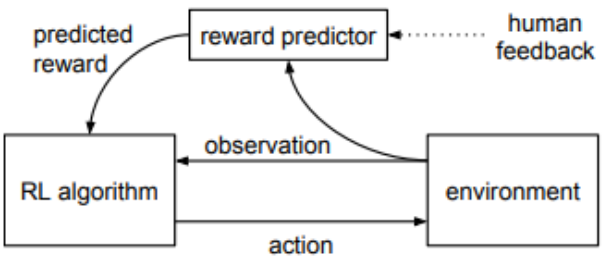
By treating an LLM as an agent in a reinforcement learning set up, where the generated tokens are the actions which are rewarded with respect to some scoring system, we will incentivize the LLM to parametrize itself in accordance with our scoring metric. This mechanism is reflected in two dominant paradigms:

- RLHF (Reinforcement Learning with Human Feedback)
- RLVR (Reinforcement Learning with Verifiable Rewards)

Table 2.1: Elements of RLVR training in an LLM

RLVR component	**Part**
Agent	LLM
Environment	input dataset and reward harness, e.g., verifier
State	token history and prompts till time t
Action	generation of tokens

2.1.1 RLHF and RLVR



Courtesy: Deep Reinforcement Learning from Human Preferences[11]

Christiano et al[11]. explored learning a reward function from human feedback and optimizing it. Although this has been grazed upon before, the authors emphasize and tackle scaling the same to modern deep RL. The reward predictor is independently trained using human feedback (e.g., through comparisons of preferable trajectory pathways in Atari), and the agent(LLM) tries to maximize the reward.

RLVR is an analogue of RLHF, except only the reward predictor functions like a verifier. The verifier can be programmatic (symbolic verifier), or we can deploy another larger, sturdier LLM to judge the reasoning and the final output.

Policy gradient uses some performance measure $J(\theta)$ of the policy (where θ implicitly translate to the weight parameters of our LLM model). Using gradient ascent we update the parameters:

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{2.1}$$

where α is the learning rate and $\widehat{\nabla J(\theta_t)}$ is a stochastic estimator for approximating the gradient of $J(\theta_t)$.

Guo et al. [12] widely popularized the usage of modern RLVR to improve LLMs. The noteworthy benchmarks set by Deepseek as a result were also corroborated by Wen et al.[13] and a CoT-pass@k metric was introduced that evaluated not only the final output but also the reasoning steps produced alongside.

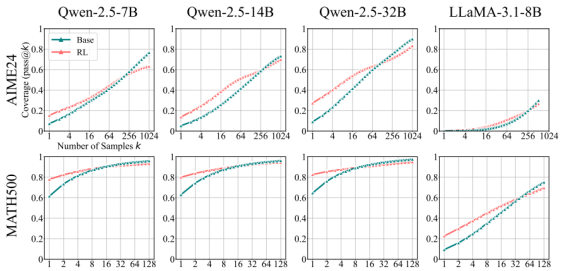
2.1.2 The problem with RLVR

Despite the flourishing advancements via RLVR, a growing body of research points to issues within this domain. The primary motivator for these concerns is the fact that post-RLVR trained models improve the pass@1 score, but the same is not reflected in pass@k for higher k's when compared to the base models. This phenomenon was first observed by Shao et al. [14] in the seminal paper that introduced GRPO via the release of DeepSeekMath 7B. This query was more systematically dealt by Yue et al.[15] who showed that the pass@k metric of the base model goes up much more rapidly than the RLVR-trained model. In fact for a large k, the base model matches or even surpasses the RLVR counterpart entirely. These findings motivate their proposal that all correct reasoning paths already exist within the base model, and RLVR is just a sampling efficiency tool that functions at the cost of lowering the overall reasoning mettle of the model. Wang et al. [16] published results that appear to corroborate this hypothesis: they showed significant improvements in RLVR by restricting policy gradient updates to tokens with the top 20 percent entropy, calculated as:

$$H_t := -\sum_{j=1}^{V} p_{t,j} \log p_{t,j}, \text{ where } (p_{t,1}, \ldots, p_{t,V}) = \boldsymbol{p}_t = \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{q}, \boldsymbol{o}_{<t}) = \text{Softmax}\left(\frac{\boldsymbol{z}_t}{T}\right). \tag{2.2}$$

where π_θ is the LLM with the weight parameters θ, $\mathbf{q}$ is the input query and $\boldsymbol{o}_{<t} = (o_1, o_2, ..., o_t - 1)$ are the previously generated tokens. Shao et al.[17] also showed that RLVR can elicit substantial

mathematical aptitude even with "spurious rewards" that have no, or even a negative correlation with the correct answer, further obfuscating the mechanics behind it. Contradictory findings have also emerged in the literature e.g., Liu et al.[18] who have deployed a modified RLVR(ProRL) that uses KL divergence control, reference policy resetting, etc. It was found that ProRL models outperformed base models in pass@k metric, and even when the base model failed regardless of the number of attempts. However for math benchmarks they have maintained the regressive pass@k scores for the RLVR-tuned models. Chen et al.[19] also showed marked improvements in pass@k scores compared to SFT models. No definitive theory has been posited yet which reconciles these contradictory findings. We seek to explore further comparisons between base and RL-models.



Courtesy: Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? [15]

2.1.3 PPO: Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a prominent policy gradient method in RL. Introduced by OpenAI[20] in 2017, PPO strikes a balance between ease of implementation, sample complexity, and wall-clock time by preventing destructively large policy updates. Traditional policy gradient methods often suffer from high variance and destructive step sizes. If a policy update is too large, the agent's performance can collapse, making recovery difficult. PPO solves this by introducing a **surrogate objective function** that constrains or "clips" the policy update size, ensuring the new policy does not drift too far from the old one.

PPO optimizes the following clipped surrogate objective function at each iteration:

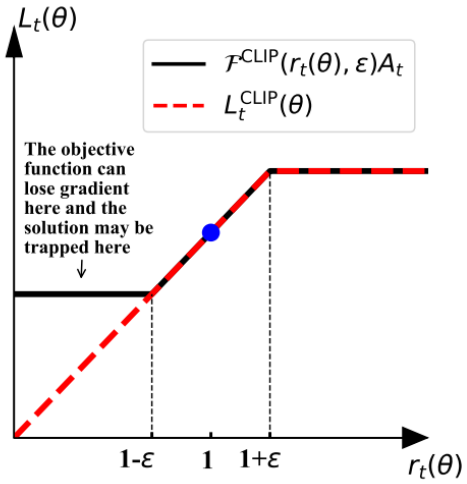
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.3}$$

Where the individual components are defined as follows:

- θ: The vector of policy parameters being optimized.
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$: The probability ratio between the new policy and the old policy.
- $\hat{A}_t$: The estimated advantage function at time step t, which measures how much better an action is compared to the average action in that state; $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \approx G_t - V^\pi(s_t)$
- ϵ: A hyperparameter (typically set to 0.1 or 0.2) that dictates the clipping threshold boundary.

The min operator acts as a pessimistic bound. It ensures that the objective does not increase beyond a certain point when the policy change would drastically improve the reward, discouraging excessively large updates. But PPO also thrusts the following challenges during implementation:

- Poor Exploration in Sparse-Reward Environments:
  The environment must provide immediate and dense feedback (rewards). If not, the PPO policy collapses. The agent gets stuck in a looping cycle of local optima because the clipping mechanism restricts it from making bold, exploratory deviations away from its poorly performing current policy.[21].



(a) $A_t > 0$

  Reference Courtesy: Truly Proximal Policy Optimization [21]

- Additional value network that inflates the VRAM usage and blows up the memory requirements and the computational costs

- Complex optimization dynamics: When code-level optimizations (such as value loss clipping, gradient clipping, and observation normalization) are stripped away, PPO's performance degrades catastrophically. The mathematical optimization itself is highly unstable.[22]

2.1.4 GRPO: Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) is a memory-efficient reinforcement learning algorithm specifically engineered for post-training Large Language Models (LLMs) [14]. Introduced as a core mechanism behind the scaling of deep mathematical and logical reasoning models, GRPO alters the policy-gradient workflow by fundamentally changing how baseline reward signals are computed. In traditional actor-critic reinforcement learning (such as standard PPO), a training run requires keeping a companion **Critic/Value network** alive in GPU memory alongside the target **Actor/Policy model**. This critic network—which is often identical in parameter scale to the base LLM—estimates a baseline value to compute an action's absolute advantage. This architecture heavily inflates VRAM overhead during optimization.

GRPO completely eliminates the explicit value network. Instead of learning an absolute value heuristic, GRPO derives baseline statistical scores on-the-fly from the joint performance of a candidate output's own sibling variations.
For an input query q, GRPO samples a group of G independent candidate completions directly from the old policy model:

$$\{o_1, o_2, \ldots, o_G\} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q) \tag{2.4}$$

Each response o_i is evaluated by a reward mechanism (such as a programmatic code test sandbox or an outcome verifier) to yield a raw scalar score r_i. The relative advantage A_i for each response is then standardized using a dynamic z-score normalization over the sampled peer group:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_G\})}{\text{std}(\{r_1, r_2, \ldots, r_G\})} \tag{2.5}$$

Even if all outputs within a batch perform poorly, outstanding marginal steps still receive strong positive reinforcement signals.

2.1.5 The GRPO Objective Function

The parameter space θ of the policy is updated by maximizing the clipped relative advantage objective, regularized with an explicit penalty ensuring close alignment to a baseline reference policy π_{ref}:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\left(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}A_i,\ \text{clip}\left(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}, 1-\epsilon, 1+\epsilon\right)A_i\right) - \beta\mathbb{D}_{\text{KL}}(\pi_\theta \parallel \pi_{\text{ref}})\right] \tag{2.6}$$

where ϵ represents the PPO-style clipping bounds to restrain massive policy adjustments, and β is the regularization coefficient managing Kullback-Leibler (KL) divergence.

2.2 LLM evaluation metrics: pass@k, maj@k, rm@k

When evaluating Large Language Models (LLMs) on mathematical benchmarks like GSM8K or code generation datasets, models generate multiple stochastic completions for a single problem. To evaluate these generative distributions, researchers utilize specific sampling metrics: **pass@k**, **maj@k** (Majority Voting), and **RM@k** (Reward Model Selection).

2.2.1 The Pass@k Metric (pass@k)

The pass@k metric measures the probability that at least one correct solution is generated when sampling k candidate outputs from the model.

To evaluate this efficiently without massive variance, (Chen et al.[23])we sample a larger pool of n total completions ($n \geq k$) and calculate the expected success rate combinatorially. If c out of n generated samples are correct, the unbiased estimator is defined as:

Algorithm 1: Group Relative Policy Optimization

Sample prompt q Generate G responses:

$$o_1, \ldots, o_G$$

Compute rewards:

$$r_1, \ldots, r_G$$

Compute mean reward:

$$\bar{r}$$

Compute standard deviation:

$$\sigma_r$$

Compute normalized advantages:

$$A_i = \frac{r_i - \bar{r}}{\sigma_r + \epsilon}$$

Compute policy ratios. Evaluate clipped objective. Apply KL penalty. Update policy parameters.

$$\text{pass@}k = \mathbb{E}_{problems}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{2.7}$$

where:

- n is the total number of samples generated per problem.
- c is the number of correct samples within the pool of n.
- k is the number of evaluations allowed per problem.

Note: If $n - c < k$*, then* $\binom{n-c}{k} = 0$*, yielding a pass rate of* 1.0*.*

2.2.2 The Majority@k Metric (maj@k)

The maj@k metric (often referred to as Self-Consistency) evaluates model accuracy under a democratic ensemble setup. For a given question, k total samples are generated. The final answer selected is the one that appears most frequently among the k paths.

Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of k generated reasoning paths, and let $A(s_i)$ map a reasoning path to its final parsed answer. The selected answer $\hat{a}$ is defined by:

$$\hat{a} = \arg\max_a \sum_{i=1}^{k} \mathbb{I}\left(A(s_i) = a\right) \tag{2.8}$$

Where $\mathbb{I}(\cdot)$ is the indicator function. The problem is scored as correct (1) if $\hat{a}$ matches the ground-truth answer, and incorrect (0) otherwise.

2.2.3 The Reward Model@k Metric (RM@k)

The RM@k metric evaluates a Verifier or Reward Model (RM) setup. The LLM outputs k answers. A secondary scoring model (the Reward Model) evaluates all k candidates and selects the highest-scoring candidate path as the final output.

Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of k generated outputs, and let $\mathcal{R}(s)$ be the scalar score assigned to sample s by the Reward Model. The chosen solution s^* is defined as:

$$s^* = \arg\max_{s \in S} \mathcal{R}(s) \tag{2.9}$$

The problem is scored as correct if the parsed answer of s^* matches the ground-truth answer.

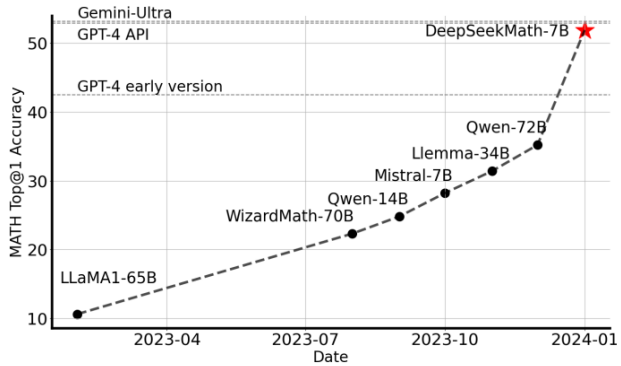
2.3 Math Benchmarks

Evaluating mathematical reasoning in LLMs has come a long way from run-of-the mill next token prediction algorithm to reinforcement learning fueled computations. We will now proceed to track the progress of frontiers in mathematical advancements in this regard:

GSM8K: Introduced in 2021 by Cobbe et al.[24], this dataset of 8.5K school grade math problems had been used as the primordial math benchmark for natural language problems in English at a time when mathematical reasoning was still nascent for LLMs. GPT-3(175B) set a score of 35% test accuracy. Several variations of this datasets have also been constructed.

MATH: Developed by Hendrycks et al.[25], this dataset comprises 12,500 mathematics problems with scholastic competition difficulty. Each question in the dataset has a step by step solution which can teach models step by step thinking and derivation. An accompanying pretraining dataset has also been drafted which can help the model learn fundamentals.

As mentioned earlier, DeepSeekMath 7B cinched an impressive 60.9% test accuracy on this set, bringing it on par with other reasoning models in vogue whilst debuting GRPO technique. [14]



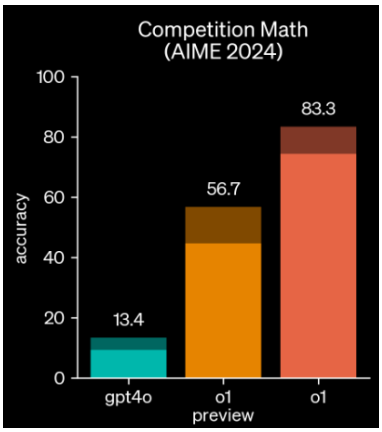
Courtesy: DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models[14]

AIME: American Invitational Mathematics Examination is a prestigious invitation-only math competition of 15 questions organised for American high-schoolers[26]. Qwen2.5-Math-7B-Instruct

was able solve 21/30 questions from the AIME 2024[27] (authors have graded this score on an rm@64 metric). In contrast, Claude3 Opus, GPT-4 Turbo, and Gemini 1.5 Pro solved only 1 or 2 questions out of 30[28].

Additionally, Qwen2.5-Math-7B-Instruct surpassed all the models in 2024 by scoring a whopping 88.1% accuracy in the MATH benchmark. Qwen2-Math-72B-Instruct also scored 96.7% accuracy in GSM8K benchmark.

Subsequently, OpenAI o1 scored an 83% accuracy rate on the AIME '24 dataset, showing a marked improvement from its predecessors.



o1 greatly improves over GPT-4o on challenging reasoning benchmarks. Solid bars show pass@1 accuracy and the shaded region shows the performance of majority vote (consensus) with 64 samples.

Courtesy: Learning to reason with LLMs[29]

Putnam and Advanced Olympiads: There is a glaring fundamental limitation of the benchmarks described above, especially in the pursuit of true mathematical proficiency: a correct answer doesn't guarantee coherent reasoning steps. Proving a theorem requires consistently derived steps and rigorous axiomatic thinking, albeit verification of a correct answer becomes more tricky. Shao et al(2025).[30], for example, have expounded on using self verification methods to tackle this and their model DeepSeekMath-V2, exhibits exemplary theorem-proving abilities: It has secured gold-level scores on IMO 2025 and CMO 2024 and an almost perfect score of 118/120 on Putnam 2024 with scaled test time compute.

Table 2.2: Chronological breakdown of milestone models setting mathematical benchmarks.

Year	**Model Group**	**Key Paradigm**	**Primary Benchmark**	**Performance Highlight**
2021	GPT-3 [24]	Text Verifiers	GSM8K	Set the baseline for grade-school multi-step text reasoning.
2024	DeepSeekMath [14]	GRPO Reinforcement	MATH	7B model achieved 60.9%, matching massive proprietary models.
2024	Qwen2.5-Math [28]	Tool-Integrated Reasoning (TIR)	AMC / AIME	Iterative self-improvement
2024	OpenAI o1 [29]	Test-Time Compute	AIME	Jumped AIME scores from 13% to 83%
2025	DeepSeek-V2 [30]	Self-Verification	Putnam 2024	Scored 118/120 on Putnam via scaled test-time proof verification.

Chapter 3

Methodology

3.1 Background

We will assess the accuracy of all the base models on the GSM8K datasets and compare the results with their RLVR-tuned counterparts. The original papers[14] have evaluated the efficacy of GRPO on GSM8K and MATH datasets for mathematical questions in the English language, and in the Chinese language, they evaluated their models on CMATH[31](a dataset comprising 1.7k mathematics problems from the elementary school level. In 2023, only GPT-4 managed to score above 60% across all the grade levels listed within the dataset. DeepSeekMath-Base had a 71.7% accuracy rate and the RLVR-tuned version DeepSeekMath-RL had an accuracy rate of 88.8%).

Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?	Natalia sold 48/2 = <<48/2=24>>24 clips in May. Natalia sold 48+24 = <<48+24=72>>72 clips altogether in April and May. #### 72

Problem

Find the number of integers less than or equal to 100 that are equal to $a + b + ab$ for some choice of distinct positive integers a and b.

Solution 1

Let x be equal to $a + b + ab$. Adding 1 to both sides, we get $a + b + ab + 1 = (a + 1)(b + 1) = x + 1$. Because we know a and b have to be positive, this means that $x + 1$ cannot be prime. We have 25 primes less than 100, but we have to also count 101 since $x = 100$ is still in the range of 1 to 100. Another thing we can notice is that perfect squares of primes don't work either. The primes whose squares are less than 100 are 2,3,5, and 7. Therefore, the answer is $100 - 26 - 4 = \boxed{070}$.

- ChickensEatGrass

A question-answer example from the GSM8K dataset(top) and 2026 AIME-I, problem 4(bottom)
(**Courtesy:** gneubig/aime-1983-2024 dataset and AoPs respectively)

Grade	Problem in Chinese	English translation	Answer	#Steps	#Digits
1	商店里有 9 个蛋糕，卖出去 5 个，还剩多少个?	There are 9 cakes in the store. If 5 are sold, how many are left?	4	1	1
2	公交车上原有 32 人，到站后上来 15 人，又下车 4 人。现在公交车上有多少人?	Originally there were 32 people on the bus, 15 more got on at the next stop, and 4 got off. How many people are on the bus now?	43	2	2
3	某商店里手套 12.4 元一副，帽子 35.7 元一顶。买一副手套和一顶帽子一共要多少钱?	A pair of gloves costs 12.4 yuan and a hat costs 35.7 yuan in a store. How much does it cost to buy a pair of gloves and a hat together?	48.1	1	3
4	一只箱子里装有 6 只脚的蟋蟀和 8 只脚的蜘蛛，它们共有 66 只脚。箱子里有蟋蟀多少只?	A box contains crickets with 6 legs and spiders with 8 legs. Together they have 66 legs. How many crickets are in the box?	3	5	2
5	一根竹竿插入水中，入水部分长 5/14 米，入泥部分 1/14 米，露出水面 3/14 米。这根竹竿长多少米?	A bamboo pole is inserted into the water, with 5/14 meters of it submerged, 1/14 meters in the mud, and 3/14 meters exposed above the water surface. How long is the bamboo pole in total?	5/14	2	3
6	张老师到银行存款 4500 元，年利率是 2.25%，扣除 20% 利息税，一年后取回本息多少元?	Teacher Zhang deposits 4500 yuan in the bank at an annual interest rate of 2.25%. After deducting 20% interest tax, how much will he get back in one year?	4581	4	4

Examples of all the six grade levels of questions in CMATH.[31]

3.2 RLVR-tuning the base models:

Traditional fine-tuning adjusts every single weight in a massive model, which is incredibly expensive, slow, and requires massive amounts of computing power (like multiple high-end GPUs). Since this is not feasible in our case. We will deploy the aid of techniques from Parameter Efficient Fine-Tuning(PEFT)

3.2.1 What is Fine-Tuning

Introduced in 'Improving Language Understanding by Generative Pre-Training'[32], fine-tuning a pre-trained model involves sharpening its accuracy on a more specific subset of the dataset on which it was trained. Mathematically, while pre-training, we aim to solve the following optimization problem:

$$\theta' = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(y, f_\theta(x))\right] \tag{3.1}$$

Let $\mathcal{D}$ represent the target domain dataset consisting of input-output pairs (x, y), and $\mathcal{L}$ represent the objective loss function. However, finetuning proceeds to find:

$$\theta'' = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}'} \left[\mathcal{L}(y, f_\theta(x))\right] \tag{3.2}$$

where θ'' is a small perturbation from the original parameter θ' of the model, such that error is minimized on the new dataset $\mathcal{D}'$.
While highly effective, updating billions of parameters introduces massive computational costs, high memory consumption, and risks (e.g.,*catastrophic forgetting*) of the model's core capabilities.

3.3 Parameter-Efficient Fine-Tuning (PEFT)

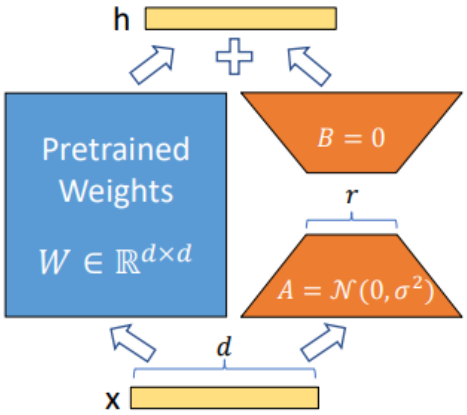
To mitigate the resource constraints of full fine-tuning Parameter-Efficient Fine-Tuning (PEFT) methods freeze the original weights and only update a small fraction of specialized parameters. There are several paradigms of PEFT manipulation:

- **Low-Rank Adaptation (LoRA):** Decomposes weight changes into small, low-rank matrices to reduce the number of trainable variables drastically.
- **Prefix-Tuning:** Appends trainable continuous vectors (virtual tokens) directly to the keys and values across attention layers.
- **Prompt Tuning:** Prepends learnable task-specific embedding vectors directly to the input sequence prompt.

Out of these, we will focus on LoRA which we have used to maximize efficiency in RLVR-tuning our code.

3.4 LoRA[33]

As the number of parameters skyrocket, computing the optimal parameters becomes increasingly computationally prohibitive. To overcome this, we have used **Low-Rank Adaptation (LoRA)** and its successor, **Quantized LoRA (QLoRA)**, which are state of the art training techniques especially when there are limitations on the training hardware.



We only update the matrices A and B. **Courtesy:** LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

LoRA freezes the training weights of the model and "injects trainable rank decomposition matrices (e.g., A and B)" inside every layer in the transformer architecture. This greatly diminishes the quantity of trainable parameters for tasks further down the line. Motivated by the low intrinsic dimension of over-parametrized models[34][35] it was hypothesized that the weight change during model adaptation also has a low rank which has lent itself to the conception of Low-Rank Adaptation (LoRA) approach. LoRA has the following benefits:

- Given a single pre-trained model, several LoRA modules which specialize in different variety of sub-tasks can be elicited. By simply switching between the different matrices A,B we can significantly bring down the storage size and the switching over-head
- Since we only optimize for smaller matrices, the barrier to hardware entry is considerably decreased
- This simple linear design completely eliminates inference latency since the trainable matrices can be conveniently merged with the frozen weights during inference time.
- LoRA is orthogonal to other fine-tuning methods. This allows us to use it in tandem with other techniques to maximize efficiency

For the purposes of this discourse we will be relying on the terminology and references introduced by Vaswani et al.[7]
For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA freezes W_0 and parametrizes its update ΔW by decomposing it into two lower-rank matrices $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$, where the rank satisfies $r \ll \min(d, k)$.

3.4.1 Forward Pass Formulation

For a given input vector x, the modified forward pass layer calculation is mathematically expressed as:

Table 3.1: Symbols

Entity	**Notation**
Input and output dimension size of the transformer layer	d_{model}
Query projection matrix	W_q
Key projection matrix	W_k
Value projection matrix	W_v
Output projection matrix	W_o
Rank of the LoRA module	r
Original weights of the model	W_0 or W
Accumulated gradient update (adaptation)	ΔW

$$h = W_0 x + \Delta W x = W_0 x + \frac{\alpha}{r} BAx \tag{3.3}$$

A dense neural network performs multiple matrix multiplication calculation across its many layers. Additionally these matrices are full rank generally. However it has been shown that pretrained models have a low intrinsic rank. Consequently, this has led us to surmise that updates to the weights during adaptation also share the same feature.
In equation 3.3, W_0 is frozen during training; it is not subjected to any gradient updates, unlike the matrices A and B which are trainable. Furthermore:

- A is a matrix initialized via a Gaussian distribution.
- B is a matrix initialized to zero, ensuring $\Delta W = 0$ at the start of training.
- α is a constant scaling hyperparameter that controls the adapter's influence.

3.5 QLoRA: Quantized LoRA

Introduced by Dettmers et al.[36], this modification of QLoRA reduces the memory usage entailed in finetuning a model while substantially preserving the finetuning task performance. Generally, model weights require 16-bit or 32-bit floating point precision, which can be incredibly taxing on memory as the model size goes up. QLoRA can bring down the average memory needed to finetune a 65 billion parameter model from >780GB to roughly less than 48GB. To see how this unfolds, we first ask:

3.5.1 What is quantization?

Typically, this is the process of converting an input from a more informational representation to a less informational representation,[36] for example converting from 32-bit floats to 8-bit Integers.

Since we want to maximize the usage of the entire range of information afforded to us by the lower range representation, we rescale the input datatype into the target datatype range by normalizing by the absolute maximum of the input elements, which usually exist in tensor form. Below, we have demonstrated the quantization of 32-bit Floating Point (FP32) tensor into an Int8 tensor with range[127, 127]:

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}) \tag{3.4}$$

where c is the quantization constant (or quantization scale). The inverse of this operation is dequantization. Mathematically:

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}} \tag{3.5}$$

However if there is an outlier amongst the tensor of input values, i.e., if there is an input of considerable magnitude, a lot of quantization bins (certain bit combinations) will remain empty. To overcome this problem, we partition the input tensor into independently quantized blocks each equipped with its own quantization constant c. We split the input tensor $\mathbf{X} \in \mathbb{R}^{b \times h}$ into n contiguous blocks of size B by flattening the input tensor and we split the linear array into $n = (b \times h)/B$ blocks. These blocks are consequently independently quantized. in accordance with reference equation (3.4), hence creating a quantized tensor and n quantization constants $c_i, i \in [n]$ QLoRA has two datatypes based on the use case:

- Storage: Low precision (e.g., 4-bit)
- Computation: e.g., BFloat16 (Brain Floating Point 16-bit, developed by Google Brain) Everytime a QLoRA weight tensor is used, it is first dequantized to BFloat16 from 4-bit representation. Subsequently necessary matrix multiplication is performed in 16 bit.

QLoRA seeks to achieve 4-bit fine tuning via mainly two strategies:

- 4-bit NormalFloat (NF4)
- Double Quantization

3.5.2 4-bit NormalFloat (NF4):

We use quantile quantization, an information-theoretically optimal data type that splits in such a way that each quantization bin has roughly equal number of elements from the input tensor[37]. This method works by estimating the quantile of the input tensor elements through an empirical CDF. However the approximate nature of these estimation algorithms can involve large errors, especially on the outlier values

Luckily, pretrained Neural Networks have weights which are manipulated to be normally distributed with mean 0 and standard deviation σ. We scale all these normal distributions so that we get the

weight values within the range [-1, 1], because we intend to fit them into the range of our target datatype. The algorithm followed henceforth is referenced is pseudo-coded below.[36]

Algorithm 2: Information-Theoretically Optimal k-bit NormalFloat (NFk) Quantization

Algorithm: Information-Theoretically Optimal k-bit NormalFloat (NFk) Quantization

Input: Bit-width k

Input weight tensor $W \in \mathbb{R}^{n \times m}$

Quantile function $Q_X(\cdot)$ of the standard normal distribution $\mathcal{N}(0, 1)$

Output: Asymmetric NFk data type array $q = \{q_0, q_1, \ldots, q_{2^k-1}\}$

Quantized weight tensor q^W

Quantization scale constant c

// Step 1: Construct the asymmetric quantile sets to ensure an exact 0 representation

Initialize empty sets for negative and positive ranges: $\mathcal{Q}_{\text{neg}} \leftarrow \emptyset$, $\mathcal{Q}_{\text{pos}} \leftarrow \emptyset$

// Estimate 2^{k-1} quantiles for the negative distribution boundary

for $i = 0$ **to** $2^{k-1} - 1$ **do**

  $q_{\text{neg},i} = \frac{1}{2}\left[Q_X\left(\frac{i}{2^k+1}\right) + Q_X\left(\frac{i+1}{2^k+1}\right)\right]$

  $\mathcal{Q}_{\text{neg}} \leftarrow \mathcal{Q}_{\text{neg}} \cup \{q_{\text{neg},i}\}$

// Estimate $2^{k-1} + 1$ quantiles for the positive distribution boundary

for $i = 2^{k-1}$ **to** 2^k **do**

  $q_{\text{pos},i} = \frac{1}{2}\left[Q_X\left(\frac{i}{2^k+1}\right) + Q_X\left(\frac{i+1}{2^k+1}\right)\right]$

  $\mathcal{Q}_{\text{pos}} \leftarrow \mathcal{Q}_{\text{pos}} \cup \{q_{\text{pos},i}\}$

// Step 2: Unify sets and normalize the data type range into $[-1, 1]$

$\mathcal{Q}_{\text{unified}} \leftarrow \mathcal{Q}_{\text{neg}} \cup \mathcal{Q}_{\text{pos}}$ // Combines sets and implicitly removes duplicate 0

$q \leftarrow \text{Sort}(\mathcal{Q}_{\text{unified}})$ // Yields array q of size 2^k

$q \leftarrow \frac{q}{\max(|q|)}$ // Normalize values exactly into $[-1, 1]$

// Step 3: Absolute Maximum Rescaling and Weight Quantization

$c = \max(|W|)$ // Compute the weight scaling constant

$W_{\text{norm}} = \frac{W}{c}$ // Normalize input weight tensor into $[-1, 1]$

$q^W = \text{MapToNearest}(W_{\text{norm}}, q)$ // Quantize using the matched data type bins

return q, q^W, c

3.5.3 Double Quantization

The aforementioned quantization constants also contribute to significant memory overhead. For example, if we are using a blocksize of 64 for 32 bit constants, there will be an additional 0.5 bits per parameter in the memory. To mitigate this issue, we in turn quantize these constants as well.

We call the quantization constants derived in the first iteration c_2^{FP32}. These now form the inputs of the the second quantization iteration which now are morphed into c_2^{FP8}. The quantization constants of these, in turn, are c_1^{FP32}. For the second quantization, we generally resort to 8 bit floats and a blocksize of 256.

3.5.4 Paged Optimizers

Additionally, QLoRA can also leverage **Paged Optimizers** which involves automatic page to page transfers between CPU and GPU to prevent any computational bugs that might arise as a result of the GPU running out of memory. This feature is analogous to the memory paging between CPU RAM and the disk.

3.5.5 QLoRA

QLoRA for a single linear layer in the quantized base model with a single LoRA adapter is defined as below:

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{A}^{\text{BF16}}\mathbf{B}^{\text{BF16}}, \tag{3.6}$$

where doubleDequant$(\cdot)$ is defined as:

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}} \tag{3.7}$$

W has been split into block sizes 64 for higher quantization precision and 256 for c_2 for memory preservation. Moreover, given the error E, for the parameter updates, instead of computing $\frac{\partial E}{\partial W}$, we compute the gradients only on the adapters, i.e., $\frac{\partial E}{\partial A}$ and $\frac{\partial E}{\partial B}$ in accordance with LoRA. Since the computation of $\frac{\partial E}{\partial A}$ and $\frac{\partial E}{\partial B}$ involves computing $\frac{\partial X}{\partial W}$ of the layer that succeeds it, we leverage the above equation to dequantize $\mathbf{W}^{NF4}$ from storage type to computational data type $\mathbf{W}^{BF16}$ to find the necessary gradient.

Chapter 4

Experiment

We will RLVR-tune four base models and compare the efficiency of the base models and their RLVR-tuned versions on GSM8K dataset.

4.1 RLVR-tuning the models:

We select the following models for comparison:

- models/qwen3_8B
- Qwen/Qwen2.5-Math-7B
- meta-llama/Llama-3.1-8B
- google/gemma-3-12b-pt

We RLVR-tune each of these models in accordance with their pretraining background to sharpen the RLVR-tuning accuracy as much as possible.

4.1.1 Memory Management:

Since RLVR-tuning can be extremely taxing on the memory, in order to avoid OOM errors and computational hiccups, we borrow the `LoraConfig` from `peft` library.
In order to further optimize the space utilized we make use QLoRA techniques using `BitsAndBytesConfig` from `transformers`. We also use Scaled Dot-Product Attention (SDPA).

4.1.2 GRPO:

For the GRPO training, we rely on `GRPOConfig` and `GRPOTrainer` from `trl`.

- `num_generations = 4`: This is preferably a power of 2 for hardware alignment reasons. A lower generation choice will result in a highly fluctuating variance resulting in highly unstable mathematical updates that can result in degradation of model performance.

- `gradient_accumulation_steps = 4`: Instead of updating the model's weights after every single step (which requires a massive burst of VRAM to calculate gradients), the GPU performs 4 separate micro-batches back-to-back, tracking and adding up (accumulating) the gradients mathematically. It combines them all and performs one structural weight update. Effective Batch Size becomes 4 (batch size) $\times$ 4 (accumulation) = 16 prompts per full update.
- The Effective batch size must be divisible by `num_generations = 4`. The library ensures that groups of generations are not split across updates.

4.1.3 Qwen/Qwen2.5-Math-7B

Trained on advanced mathematical reasoning tasks, like CoT and Tool-integrated reasoning.[38] The base accuracy calculated was: 50% Post RLVR-tuning, it jumps to a remarkable 66.66% accuracy.

4.1.4 Qwen-3 8B:

This model[39] has the most promising performance of all four - It has a base performance of 94%. This can be explained by:

- Thinking budget allocation which can exploit computational resources adeptly during inference time.
- Pretraining on 36 trillion tokens
- Rigorous pretraining in STEM topics and coding with focus on long context data and long CoT finetuning

However, after the RLVR-tuning, the accuracy rate paradoxically tanks at: 72.72% The factors that might contribute to this anomaly could be:

- `max_completion_length` has been set at 1024 due to memory constraints. However, the final evaluation metrics show that `eval_completions/clipped_ratio: 0.2575` which indicates that 25.75% of the model's generations are being cut off prematurely. So even if the model is capable of generating the correct solution, we are bound by the prompt length restrictions
- GRPO clipping ratio may be too tight

The next two models lack the rigorous math domain enrichment, so the run-of-the-mill reward function eventually leads to a gradient collapse. To circumvent this, we modify the reward function slightly, and assign a modicum of points to the correct format, in addition to the scores for the correct answer.

4.1.5 Llama-3.1-8B

Meta's open weight model, it is optimized for multilingual conversational uses[40]. It has a base accuracy of: 3%
After RLVR-tuning the accuracy massively improves: 12.27%

4.1.6 Gemma-3-12B-pt

: Introduced by Google DeepMind, some versions of this family are equipped to run on standard consumer grade hardware.[41] The base accuracy rate calculated was 2%, and the RLVR-tuning resulted in an accuracy of: 34%

Table 4.1: Accuracy rates of Base and RLVR-tuned LLMs

Models	**Base (%)**	**Post RLVR(%)**
Qwen3-8B	94	72.72
Qwen2.5-Math-7B	50	66.66
Llama 3.1 8B	3	12.27
Gemma-3-12B-pt	2	34

Chapter 5

Conclusions

This dissertation explores the logistics of RLVR-tuning a pretrained base model on mathematical capabilities while optimizing the computations given memory constraints.

We began by reviewing the basic theoretical foundations of Deep Neural Networks and outlining how Universal Approximation Theorem solidified the significance of studying these structures owing to their versatility in approximating any continuous function on a compact subset.

We then surveyed the progress of Language models starting from the unassuming N-gram language models which restrict the prediction of the next word to a fixed history window to the first Neural language model, RNN and its successor LSTM which expanded the aforementioned window and allowed us to take weighted context from the past. However, the introduction of Transformers wholly revolutionized the scene of language modeling by integrating parallelization via self attention.

Next, we described Chain-of-Thought techniques that helped the model to reason more intuitively (like humans do), by making them elucidate their process in several natural language texts, instead of an ultimate answer. We also explored self-consistency in this context and Tree-of-thought techniques for a more fine-grained approach towards step by step reasoning.

This was followed by motivating RLVR through RLHF, and by laying the ground work of reinforcement learning principles to build the heuristics of Reinforcement learning through verifiable rewards. We also discuss the growing literature regarding the shortcomings of this paradigm. Currently there is no consensus regarding the exploratory merit of RLVR and it continues to be a highly volatile area of research.

The importance of parameter efficient fine tuning in cannot be overstated especially when working in limited memory settings. Out of various techniques in this realm, we focus on LoRA, wherein we selectively perform updates only on the lower dimensional matrix decomposition ΔW which is the difference in weight parameters of the matrix under the new subset dataset. This notion is further polished by manipulating the datatype of weights when under storage or computations, in QLoRA.

We RLVR-tuned 4 pretrained base models using appropriate reward functions and custom tuned specifications. We applied the necessary constraints on memory and tackled each pretrained model keeping in mind its pretraining dataset to maximize the accuracy rate that would be possible within our given hardware limitations. We finally calculated the accuracy of the base model and their

RLVR-tuned versions and compared them, corroborating the success of RLVR in improving the performance of some large language models, however based on the degradation of performance for the strongest model, we assert that the effectiveness of RLVR depends on the model characteristics and the training set up as well.

5.1 Future Work

Building on the premise established in this dissertation, we can venture in several promising avenues that explore mathematical reasoning in large language models:

- Scaling to Larger Models: Effectiveness of GRPO on significantly larger foundation models can be explored. It is important to comprehend how reward allocation, training stability and answer sampling efficiency evolves with a growing model size.
- Advanced Reward Design: The current state of RLVR inherently relies on verifiable reward functions. Future studies can expound on hybrid reward mechanisms that combine verifiable correctness with learned reward models
- Multi-step Reasoning and Planning: Extending RLVR reliably to longer reasoning chain remains a challenge. Reward mechanisms that not only hinge on the final correctness but also the robustness of the intermediate coherence can be focused on.
- Improved parameter efficient Reinforcement Learning: Although LoRA and QLoRA substantially reduce training costs, PEFT techniques continue to evolve. Comparative studies involving DoRA, AdaLoRA, and other adaptation methods could be studied with regard to their suitability in reinforcement learning training
- Sample Efficiency and training stability: One of the reasons for the high computation overhead of GRPO is comes from the huge number of generated trajectories. If answers are more efficiently sampled, GRPO's performance will be boosted.

In conclusion, the combination of RLVR, GRPO, LoRA and QLoRA provides a practical and a scalable framework for enhancing the mathematical reasoning abilities of large language models. Continues research in reward design, sampling efficiency and optimized fine-tuning will drive the advancement of robust and competent reasoning systems, bringing it closer to a real world deployment.

References

[1] C. Stryker and E. Kavlakoglu, *What is artificial intelligence (AI)?* IBM Think, 2026. Accessed: Jun. 11, 2026. [Online]. Available: `https://www.ibm.com/think/topics/artificial-intelligence`.

[2] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. [Online]. Available: `https://cognitivemedium.com`.

[3] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: `https://cmu.edu`.

[4] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. [Online]. Available: `http://jhu.edu`.

[5] V. Nagarajan and J. Z. Kolter, *Uniform convergence may be unable to explain generalization in deep learning*, 2019. [Online]. Available: `https://arxiv.org`.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: `https://researchgate.net`.

[7] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008. [Online]. Available: `https://arxiv.org`.

[8] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24 824–24 837. [Online]. Available: `https://arxiv.org`.

[9] X. Wang et al., *Self-consistency improves chain of thought prompting in large language models*, 2022. [Online]. Available: `https://arxiv.org`.

[10] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models," in *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: `https://arxiv.org`.

[11] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, *Deep reinforcement learning from human preferences*, 2017. [Online]. Available: `https://arxiv.org`.

[12] DeepSeek-AI, *DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning*, 2025. [Online]. Available: https://arxiv.org.

[13] X. Wen et al., *Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base LLMs*, 2025. [Online]. Available: https://arxiv.org.

[14] Z. Shao et al., *DeepSeekMath: Pushing the limits of mathematical reasoning in open language models*, 2024. [Online]. Available: https://arxiv.org.

[15] X. Yue et al., *Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?* 2025. [Online]. Available: https://arxiv.org.

[16] Y. Wang et al., *Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning*, 2025. [Online]. Available: https://arxiv.org.

[17] R. Shao et al., *Spurious rewards: Rethinking training signals in rlvr*, 2025. [Online]. Available: https://arxiv.org.

[18] Z. Liu et al., *Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models*, 2025. [Online]. Available: https://arxiv.org.

[19] Y. Chen et al., *Acereason-nemotron: Advancing math and code reasoning through reinforcement learning*, 2025. [Online]. Available: https://arxiv.org/pdf/2505.16400.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. [Online]. Available: https://arxiv.org.

[21] Y. Wang, H. He, C. Wen, and X. Tan, "Truly proximal policy optimization," *arXiv preprint arXiv:1903.07940*, 2019. [Online]. Available: https://arxiv.org/abs/1903.07940.

[22] L. Engstrom et al., *Implementation matters in deep policy gradients: A case study on ppo and trpo*, arXiv Preprint, 2020. [Online]. Available: https://arxiv.org/abs/2005.12729.

[23] M. Chen et al., *Evaluating large language models trained on code*, 2021. [Online]. Available: https://arxiv.org.

[24] K. Cobbe et al., *Training verifiers to solve math word problems*, 2021. [Online]. Available: https://arxiv.org.

[25] D. Hendrycks et al., *Measuring mathematical problem solving with the MATH dataset*, 2021. [Online]. Available: https://arxiv.org.

[26] H. Veeraboina, *AIME problem set 1983-2024*, Kaggle Datasets, 2024. Accessed: Jun. 11, 2026. [Online]. Available: https://kaggle.com.

[27] AI-MO Team, *AIMO validation AIME dataset*, Hugging Face, 2024. Accessed: Jun. 11, 2026. [Online]. Available: https://huggingface.co.

[28] Qwen Team, *Qwen2-Math: Technical report on mathematical capabilities*, 2024. [Online]. Available: https://arxiv.org.

[29] OpenAI, *Learning to reason with LLMs*, OpenAI Research Blog, 2024. Accessed: Jun. 11, 2026. [Online]. Available: https://openai.com.

[30] Y. Shao et al., *Systematic exploration of advanced RLVR systems in sequence-to-sequence models*, 2025. [Online]. Available: https://arxiv.org.

[31] C. Zheng et al., *Functional evaluation of mathematical reasoning in LLMs*, 2023. [Online]. Available: https://arxiv.org.

[32] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, OpenAI Technical Report, 2018. [Online]. Available: https://openai.com.

[33] E. J. Hu et al., *Lora: Low-rank adaptation of large language models*, arXiv Preprint, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685.

[34] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, *Measuring the intrinsic dimension of objective landscapes*, arXiv Preprint, 2018. [Online]. Available: https://arxiv.org/abs/1804.08838.

[35] A. Aghajanyan, L. Zettlemoyer, and S. Gupta, *Intrinsic dimensionality explains the effectiveness of language model fine-tuning*, arXiv Preprint, 2020. [Online]. Available: https://arxiv.org/abs/2012.13255.

[36] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *Qlora: Efficient finetuning of quantized llms*, arXiv Preprint, 2023. [Online]. Available: https://arxiv.org/abs/2305.14314.

[37] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, *8-bit optimizers via block-wise quantization*, arXiv Preprint, 2021. [Online]. Available: https://arxiv.org/abs/2110.02861.

[38] Q. Team, *Qwen2.5-math technical report: Toward luminous mathematical reasoning in large language models*, arXiv Preprint, 2024. [Online]. Available: https://arxiv.org.

[39] Q. Team, *Qwen3 technical report*, arXiv Preprint, 2025. [Online]. Available: https://arxiv.org/abs/2505.09388.

[40] M. AI, *The llama 3 herd of models*, Hugging Face Model Hub, 2024. [Online]. Available: https://huggingface.co.

[41] G. DeepMind, *Gemma 3 technical report*, Google DeepMind Media, 2025. [Online]. Available: https://googleapis.com.