# GRPO Training for Mathematical Reasoning (ML Assignment 3)

**Tsinghua University**
Allan Li (李盼加乐), 2025403473

**Model:** Qwen-2.5-1.5B-Instruct | **Dataset:** GSM8K | **Algorithm:** GRPO

## Introduction

This project focuses on fine-tuning a Large Language Model (LLM) for mathematical reasoning using **Group Relative Policy Optimization (GRPO)**. GRPO is a simplified variant of Proximal Policy Optimization (PPO) designed specifically for reasoning tasks where a verifiable ground truth exists (e.g., math problems).

## 1. Algorithm and Implementation Overview

### Key Algorithm Differences

Unlike standard RLHF methods that rely on a separate Value Network (Critic) to estimate a baseline, GRPO computes advantages by sampling a **group of outputs** for the same prompt.

- **No Critic:** The baseline is the average reward of the group.
- **Relative Advantage:** An answer has a positive advantage if it performs better than the group average.
- **Objective:** We optimize a PPO-style clipped objective to increase the probability of high-reward reasoning chains while maintaining stability.

> ⓘ **Implementation Details**
>
> The implementation uses the **Qwen-2.5-1.5B-Instruct** model. Key engineering challenges addressed included:
>
> - **Memory Optimization:** Using Gradient Checkpointing and `batch_size=1` to fit training on a single RTX 4090 (24GB VRAM).
> - **Decoder Compatibility:** enforcing `padding_side="left"` to ensure correct generation during the rollout phase.
> - **Dataset:** A subset of the **GSM8K** dataset was used to demonstrate the emergence of Chain-of-Thought (CoT) reasoning.

# 2. Implementation Details

> **Code Reference**
>
> The complete implementation logic can be found in the `grpo_homework.py` file within the repository. The implementation follows the 5-step pipeline defined in the assignment structure.

## Core Components (The 5 TODOs)

The implementation is broken down into five distinct logical blocks that mirror the GRPO training cycle described in the DeepSeekMath paper:

### 1. Prompt Tokenization

- **Objective:** efficient preparation of input tensors for the LLM.
- **Implementation:** We utilize the Hugging Face tokenizer to convert GSM8K prompts into input IDs.
- **Key Detail:** Crucially, we enforce **left-padding** (or dynamic padding via the collator) rather than the default right-padding. This is essential for Decoder-only architectures (like Qwen/Llama) to ensure the position embeddings align correctly during the generation phase.

### 2. GRPO Advantage Computation

- **Objective:** Calculating the reinforcement signal without a Value Network (Critic).
- **Implementation:** Following the GRPO formulation, we compute the **Group Relative Advantage**. For a group of outputs sampled from the same prompt, the advantage for output $i$ is calculated by normalizing its reward against the group's mean and standard deviation.
- **Paper Reference:** This implements the core innovation of GRPO—replacing the computationally expensive Critic model with the group average baseline, significantly reducing VRAM usage.

### 3. PPO Policy Loss

- **Objective:** Updating the model weights to maximize expected reward while maintaining training stability.
- **Implementation:** We calculate the probability ratio between the new policy and the reference policy (the model state before the update step). We apply the standard **PPO**

**Clipping** mechanism (clip $\epsilon = 0.2$) to prevent the policy from updating too drastically in a single step.

### 4. Log-Probability Computation

- **Objective:** Extracting token-level probabilities for the loss function.
- **Implementation:** Since the model outputs logits for the *entire* vocabulary, we use `torch.gather` to efficiently extract only the log-probabilities of the specific tokens that were actually generated. This requires shifting the logits by one position to align predictions with the ground-truth "next tokens."

### 5. The Training Loop (Rollout & Update)

- **Objective:** Orchestrating the full RL pipeline.
- **Implementation:** The loop executes the following cycle for each batch:
  1. **Rollout:** Generate $G = 4$ completions per prompt (Temperature sampling).
  2. **Evaluation:** Score completions against ground truth (1.0 for correct, 0.0 for incorrect).
  3. **Advantage:** Compute group-relative advantages (Step 2).
  4. **Update:** Compute loss and perform backpropagation.
- **Optimization:** To fit the Qwen-1.5B model on a single GPU, **Gradient Checkpointing** was enabled, and the batch size was strictly limited to avoid OOM errors during the generation phase.

## 3. Training Hyperparameters

The following hyperparameters were selected to balance training stability with the hardware constraints of a single RTX 4090 (24GB VRAM).

| Hyperparameter | Value | Justification |
|---|---|---|
| **Model** | Qwen-2.5-1.5B | Optimized for instruction following and math (CoT). |
| **Learning Rate** | `5e-6` | Significantly lower than SFT rates to prevent policy collapse. |
| **Group Size ($G$)** | `4` | **Critical:** Reduced from 8 to fit within 24GB VRAM. |
| **Batch Size** | `1` | Reduced to maximize memory available for generation. |
| **Max New Tokens** | `128` | Sufficient for step-by-step reasoning without OOM. |
| **Clip Epsilon ($\epsilon$)** | `0.2` | Standard PPO constraint to limit policy updates. |
| **Precision** | `bfloat16` | Reduced memory footprint while maintaining numerical range. |

⚠ **The Hardware Bottleneck**

The most significant constraint during implementation was GPU memory.

- **Group Size:** While GRPO theoretically benefits from larger groups (e.g., $G = 16$) to stabilize the baseline, we were forced to reduce $G$ to **4**.
- **Memory Impact:** Generating long Chain-of-Thought sequences for 8 parallel samples caused immediate CUDA Out-of-Memory (OOM) errors. By reducing $G$ to 4 and the Batch Size to 1, we successfully fit the training process into the available 24GB VRAM.

# 4. RL Training Logs & Analysis


**Training Progression**

The training curve demonstrates a strong positive learning trajectory, confirming the effectiveness of the GRPO algorithm even with limited compute resources.

## Analysis of Reward Curve

- **Initial Learning Phase (Steps 0–25):** The model exhibits a rapid ascent, improving from near-zero performance to an average reward of **~0.4** within the first 25 steps. This likely corresponds to the model learning the required output format and solving simpler arithmetic problems.
- **Exploration & Instability (Steps 25–150):** The reward stabilizes between 0.4 and 0.55, with noticeable variance in the raw data (light blue line). This high variance is expected given the small batch size ($B = 1$) and the binary nature of the reward signal (0 or 1).

- **Peak Performance (Steps 200–210):** A significant breakthrough occurs around step 200, where the smoothed reward peaks at approximately **0.68**, with raw batch rewards hitting **0.75+**.
- **Convergence:** By step 250, the model stabilizes at a reward of **~0.55**, indicating that the fine-tuned policy correctly answers more than half of the queries in the training distribution, a measurable improvement over the baseline.

---

## 5. Before/After Answer Analysis

> **Key Finding**
>
> The GRPO-tuned model demonstrates significantly improved **instruction adherence** and **arithmetic stability**. While the Base Model frequently hallucinates constraints (e.g., changing fractions) or generates excessive conversational filler, the GRPO model converges directly on the mathematical steps required.

### Concrete Example: The Election Problem (Test Case 2)

**The Prompt:**
*At 8:00, 5000 people lined up... By midday **2/5** of the people had voted... What's the number of those who had not voted by 16:00?*

| Base Model (Before) | GRPO Tuned Model (After) |
|---|---|
| **Status:** ❌ Incorrect | **Status:** ✅ Correct |
| **Logic:** "By midday (noon)... **half of them** have voted." | **Logic:** "By midday, **2/5** * 5000 = 2000 people have voted." |
| **Calculation:** $5000 \times 0.5 = 2500$ | **Calculation:** $5000 - 2000 = 3000$ remaining. |
| **Result:** 833 | **Result:** 1000 |

### Detailed Analysis

1. **Hallucination Reduction:**
   The Base Model failed immediately by hallucinating a constraint. It replaced the prompt's *"2/5 of the people"* with *"half of them"*. This is a common failure mode in untuned models where they drift into "likely" text completions rather than reasoning. The GRPO model correctly identified and used the fraction $2/5$.
2. **Reasoning Structure:**

- **Base Model:** Included conversational filler ("Since you can't have a fraction of a person...") and attempted complex rounding logic necessitated by its initial error.
- **GRPO Model:** Produced a clean, linear Chain-of-Thought (CoT). It calculated the midday count ($2000$), derived the remaining count ($3000$), and applied the next fraction ($2/3$) correctly.

3. **Output Format:**
   The GRPO model learned to stop after the answer. The Base Model often continued generating unrelated questions (e.g., *"Human: Write a Python function..."*) after the solution, whereas the GRPO model terminated generation appropriately or switched to a new question format closer to the training distribution.

## Summary of Improvements

- **Arithmetic Precision:** The model stopped approximating or rounding unnecessarily (as seen in the Base Model's "1666.6" error) and adhered to integer math where appropriate.
- **Prompt Grounding:** The tendency to ignore specific numbers in favor of generic concepts ("half") was eliminated in the test cases.