# Analyzing Emergent Chain-of-Thought Reasoning in LLMs via Reward Shaping and Prompt Engineering

Swadesh Jana, Pavel Kolev, Antonio Orvieto

June 24, 2025

## 1 Introduction

Large language models (LLMs) have demonstrated impressive capabilities across a wide spectrum of natural language processing tasks, including reasoning, question-answering, and summarization. A key factor driving this progress is the emergence of structured reasoning capabilities, such as chain-of-thought (CoT) reasoning, which enables models to generate intermediate steps before arriving at a final answer. This approach improves performance on complex tasks involving arithmetic, logic, and common sense inference by mimicking the human tendency to reason step-by-step. Wei et al. [1] first demonstrated that few-shot CoT prompting significantly enhances reasoning accuracy in large models. Later work by Kojima et al. [2] revealed that even zero-shot prompting with simple cues such as "Let's think step by step" can reliably induce this behavior, particularly in sufficiently large models.

Despite these advances, effectively training models to consistently produce robust CoT reasoning remains a significant challenge. While supervised fine-tuning (SFT) can directly teach models to generate rationales by leveraging annotated CoT traces, this approach demands costly expert-labeled data that is difficult to scale. Alternatively, reinforcement learning with verifiable rewards (RLVR) [3] uses outcome-based feedback alone to improve reasoning performance at scale, without requiring explicit rationales [4, 5]. Proximal Policy Optimization (PPO) [6] is commonly used for this purpose, supported by human preference signals or reward models trained on correctness labels [7].

Building on PPO, the Group Relative Policy Optimization (GRPO) algorithm [8] has emerged as a more efficient alternative by eliminating the need for a separate critic model and instead utilizing weak verifiers. These are lightweight heuristics that evaluate response quality based on format or correctness. Despite their simplicity, these verifiers have proven effective not only in steering the model toward correct outputs but also in fostering emergent CoT behaviors, such as self-reflection.

While GRPO has shown promising results, its underlying mechanisms remain poorly understood. More experimentation is needed to determine how reasoning behaviors emerge under this framework and to identify which components are critical for its effectiveness. Moreover, recent improvements such as Dr. GRPO [9] demonstrate the potential for enhanced optimization strategies to better support complex reasoning tasks. Another limitation of current GRPO implementations is the reliance on sparse rewards. In practice, models are typically trained using format- and correctness-based reward signals, but the lack of dense stepwise feedback limits the learning signal and makes it harder for the model to consistently improve.

To address these limitations, reward shaping has been proposed as a way to enrich sparse reward signals by embedding more informative intermediate objectives into the training process [10, 11]. Furthermore, prompt engineering techniques have proven to be a lightweight yet powerful tool to steer model behavior without modifying model parameters [1, 2]. However, how these two approaches interact with GRPO to foster reliable CoT generation has not been systematically studied.

In this work, we bridge this gap by conducting a detailed empirical analysis of how reward shaping and prompt engineering affect the emergence and quality of chain-of-thought reasoning in LLMs trained with GRPO. We investigate both the original GRPO and its improved variant, Dr. GRPO [9], and systematically evaluate their performance under different reward configurations and prompt designs. Our findings provide insights into how these techniques can enhance reasoning capability in large language models.

## 2 Preliminaries

### 2.1 Training LLMs and Chain-of-Thought Reasoning

Training large language models (LLMs) involves optimizing billions of parameters over massive corpora, making it computationally intensive. The standard paradigm consists of two major stages: pre-training and alignment. The former allows the model to learn broad linguistic and factual knowledge, while the latter tailors its outputs toward human-aligned objectives such as helpfulness, safety, and reasoning ability.

The unsupervised pre-training phase involves next-token prediction over diverse internet-scale datasets, including web text, books, code, and forums [12]. This autoregressive objective encourages models to learn rich syntactic and semantic patterns without explicit labels. While chain-of-thought (CoT) reasoning is not explicitly optimized during pre-training, the presence of procedural and explanatory text in natural corpora implicitly exposes models to intermediate reasoning structures, laying a latent foundation for CoT behaviors to emerge in large models [1].

To improve alignment with human intent and specific task formats, supervised fine-tuning (SFT) is applied to the pretrained model. This phase uses curated instruction-response datasets, often augmented with CoT exemplars, which have been shown to significantly enhance performance on multi-step reasoning tasks [2]. When training for CoT, the dataset typically consists of tuples $\mathcal{D} = \{x, c^*, y^*\}$, where $x$ is a task query, $c^*$ is an expert-annotated rationale, and $y^*$ is the correct answer. These examples provide direct supervision for both the reasoning trace and the final output, enabling the model to learn how to structure its intermediate thoughts coherently.

Reinforcement learning from human feedback (RLHF) [7] extends alignment further by using human preferences to optimize model behavior. More recently, reinforcement learning with verifiable rewards (RLVR) [3] has emerged as a scalable alternative for reasoning tasks. In RLVR, explicit rationales are no longer needed; models are fine-tuned using datasets of task-answer pairs $\mathcal{D} = \{x, y^*\}$, where the model generates its own reasoning trace. External verifiers or reward models assess the final output or intermediate steps, providing scalar feedback. In some cases, such as TinyZero [13], the ground truth answer $y^*$ can be replaced with automated verifiers to scale feedback generation.

Policy gradient methods, particularly Proximal Policy Optimization (PPO) [6], are commonly used to optimize LLMs in this setting. A trained reward model is used to assign scores to generated outputs, guiding the policy toward preferred behaviors. Applied to reasoning tasks, this framework enables models

to iteratively refine their reasoning steps, often improving coherence, consistency, and factual correctness [14].

Despite these successes, existing RL approaches face challenges in reward sparsity, verifier reliability, and the interpretability of learned reasoning strategies. These limitations motivate alternative methods, such as Group Relative Policy Optimization (GRPO), and enhancements through reward shaping and prompt engineering, which we explore in subsequent sections.

### 2.2 Group Relative Policy Optimization

In standard policy gradient approaches such as PPO [6], the advantage function is computed relative to a baseline value, typically to reduce the variance in gradient estimation. In particular in PPO, a trainable critic model estimates the value function of the current state. This can be computationally very expensive, as there are two LLMs to be trained in parallel. Group Relative Policy Optimization (GRPO) [8] addresses this by computing the agent's advantage relative to the performance of a group of peers, forming a dynamic and context-sensitive baseline.

Formally, let $\pi_\theta$ denote a stochastic policy parameterized by $\theta$, and let $A^\pi(s, a_i)$ represent the advantage function at a given state $s$ for an action $a_i$. In GRPO, multiple generations $\mathcal{G}$ are computed and the advantage for the output $i$ is computed as:

$$\hat{A}_i(s, a_i) = \frac{R_i(s, a_i) - \mathbb{E}_{j \in \mathcal{G}}[R_j(s, a_j)]}{\text{std}(R(s, a_j))}, \tag{1}$$

where $R_i(s, a_i)$ is the reward for output $i$ and $\text{std}(R(s, a_j))$ is the standard deviation of all the rewards $\{R_j | j \in \mathcal{G}\}$. For simplicity, $(s, a_i)$ is dropped. This formulation ensures that updates favor actions that outperform peer behavior within the same context, implicitly regularizing policy updates and encouraging more efficient exploration.

The PPO algorithm maximizes the following objective:

$$\mathcal{J}_{PPO}(\theta) = \frac{1}{|o|} \sum_{t=1}^{|o|} \min \left[ r A_{i,t}, \text{clip}(r, 1-\epsilon, 1+\epsilon) A_{i,t} \right], \tag{2}$$

where $r = \frac{\pi_\theta(o_t | q, o_{<t})}{\pi_{\theta_{old}}(o_t | q, o_{<t})}$, $\pi_\theta$ and $\pi_{\theta_{old}}$ are the current and old policy models, and $q, o$ are questions and outputs sampled from the question dataset and the policy $\pi_\theta$, respectively. $\epsilon$ is a clipping-related hyper-parameter introduced in PPO to stabilize training and $A_{i,t}$ is the token-level advantage for output $i$.

In GRPO, the objective is simply modified to replace $A$ by $\hat{A}$ by sampling $\mathcal{G}$ outputs from $\pi_\theta$. In order to prevent the model from reward hacking and mitigate over-optimization of the reward model, a per-token KL penalty from a reference model $\pi_{ref}$ is also added. Thus, the final objective is: maximize

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}\left[ \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \{ \min[ r \hat{A}_{i,t}, \text{clip}(r, 1-\epsilon, 1+\epsilon) \hat{A}_{i,t} ] - \beta \mathbb{D}_{KL}[\pi_\theta || \pi_{ref}] \} \right] \tag{3}$$

The KL divergence is estimated with the unbiased estimator:

$$\mathbb{D}_{KL}[\pi_\theta|\pi_{ref}] = \frac{\pi_{ref}(o_{i,t}|q,o_{<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} - \log\frac{\pi_{ref}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} - 1, \tag{4}$$

which is always positive.

### 2.3 Dr. GRPO

GRPO improved the training process characterized by a consistent increase in response length, an indication of advanced reasoning abilities such as self-reflection. However, [9] noted that this increase may be attributed to a bias inherent in GRPO. Specifically, there are two biases in the function: response-level length bias, arising from division by $|o_i|$, and question-level difficulty bias caused by dividing the centered outcome reward by std($R$). Removing these biases improves the LLM training and inference metrics significantly. The final objective can be written as:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|}\{\min[r\hat{A}_{i,t}, \text{clip}(r, 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}] - \beta\mathbb{D}_{KL}[\pi_\theta||\pi_{ref}]\}\Bigg], \tag{5}$$

where $\hat{A}_i(s,a_i) = R_i(s,a_i) - \mathbb{E}_{j\in\mathcal{G}}[R_j(s,a_j)]$

Although [9] mentioned in their work that the KL-term is insignificant and thus can be dropped, in this work the regularization term is still maintained. For simplicity, we re-write the objective function as:

$$\mathcal{J}_{GRPO}(\theta) = \mathcal{J}_{PG}(\theta) - \beta\mathbb{D}_{KL}[\pi_\theta||\pi_{ref}], \tag{6}$$

where $\mathcal{J}_{PG}(\theta)$ is the policy-gradient term given by

$$\mathcal{J}_{PG}(\theta) = \mathbb{E}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|}\{\min[r\hat{A}_{i,t}, \text{clip}(r, 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}]\Bigg] \tag{7}$$

### 2.4 Qwen-2.5 models

The Qwen2.5 model family [15] represents a state-of-the-art suite of open-source large language models (LLMs) designed to deliver strong multilingual comprehension, robust instruction-following capabilities, and efficient scalability. Several recent studies [9, 16] have successfully adopted Qwen2.5 models for research on reasoning and alignment, establishing them as a practical and competitive choice for experiments that require both performance and accessibility.

The Qwen2.5 series includes dedicated Instruct variants, which are fine-tuned specifically to follow user instructions and generate well-aligned conversational responses. These models undergo supervised fine-tuning (SFT) on curated instruction-response datasets and are further refined through reinforcement learning from human feedback (RLHF). This additional alignment training equips the Instruct models with enhanced capabilities for multi-turn dialogue, accurate question answering, and task-specific completions.

In this work, we select the 3-billion (3B) and 7-billion (7B) parameter Qwen2.5 Instruct models for our experiments. These models have a balance between computational efficiency and the ability to produce extended chain-of-thought reasoning. Thus, it makes them perfectly suited for our experiments on analyzing emergent reasoning behaviors under constrained hardware resources.

# 3 Methods

## 3.1 Entropy Regularization

During training, the entropy of an LLM typically decreases as it becomes more confident in its predictions for the training data. However, this sharpening of the output distribution can lead the model to concentrate probability mass on a narrow set of tokens, which can result in overconfident and repetitive generations. As a consequence, excessively low entropy can reduce generalization ability to other datasets, catastrophically forget previously learned tasks, and limit the model's capacity to explore alternative solutions. To mitigate this effect, we incorporate an explicit entropy regularization term into the training objective, which encourages the model to maintain a certain level of uncertainty in its token predictions, thereby promoting exploration and fostering richer chain-of-thought reasoning during learning. The updated objective can be written as:

$$\mathcal{J}_{GRPO}(\theta) = \mathcal{J}_{PG}(\theta) + \lambda \mathcal{H}(p) - \beta \mathbb{D}_{KL}[\pi_\theta || \pi_{ref}], \tag{8}$$

where $\mathcal{H}(p) = -\sum_{i=1}^{V} p_i \log p_i$ is the entropy, $p_i$ is the predicted token, V is the vocabulary size, and $\lambda$ is the entropy coefficient. Ideally, $\lambda$ is small $\sim 0.001$.

## 3.2 Prompt Engineering

Prompt engineering has been used effectively in recent literature as a method to improve the reasoning capabilities of an LLM [17, 1]. As mentioned previously, the phrase "Let's think step by step" shows significant improvement in CoT generation. We experiment with a few more prompts and direct the assistant to generate better responses. Since we attempt to solve a mathematical task, we test with some contextually appropriate prompts. The following two generic prompts are tested:

- P1: Do not randomly guess combinations of numbers and operations. Think rationally and logically.
- P2: Put each trajectory you test in a separate `<think> </think>` tag. There can be multiple `<think>` tags.

P1 is important because the model occasionally generates arbitrary combinations of numbers and operations, which often lack coherence due to the vast number of possible permutations. Even a simple setup involving four numbers and four basic operations yields hundreds of potential combinations. Without constraints, this leads to inefficient and unfocused exploration. In contrast, P2 addresses a common issue in which the model revisits previously attempted number-operation combinations or parts thereof. To mitigate this, we enforce a clear separation between different reasoning trajectories. This structural boundary

helps the model recognize and avoid redundant computations. In addition, it helps track the reasoning process and enables the use of regular expressions to extract intermediate steps, which can be used to design effective reward functions or verifiers.

### 3.3 Rewarding Schemes

Reward shaping allows for a more direct preference of model generations. By rewarding certain responses, we can direct the model to learn to generate similar responses. This reward can be in terms of reasoning, calculations, or trajectory predictions, among others. We experiment with two such rewarding schemes and investigate the model behavior while training.

The base reward scheme is based on the response format and the solution verification, as used in [8]. The ideal response format should have one or more `<think> </think>` tags with reasoning inside each tag and finally an `<answer> </answer>` tag that has the final answer to be evaluated. A correct format response receives a reward of 0.1, while a correct answer receives a reward of 1.

The first additional reward (R1) penalizes wrong solutions provided by the policy. Since the policy can sometimes come up with answers that do not equal the target number, we aim to penalize the model based on the gap between the target and the answer. Mathematically, we add a reward of $\frac{0.1}{1+g}$, where $g = |\text{target} - \text{answer}|$. This ensures even if the model is generating wrong answers, it always tries to minimize the gap from the target.

The second reward scheme (R2) we tested makes uses of the multiple think tags prompt to identify the result of each trajectory or each `<think> </think>` tag. The idea here is also to minimize the gap to the target, but on a trajectory level. We acknowledge that there can sometimes be divergence in individual trajectories. But an overall decrease in the gap is expected. Let the results be numbered as $\{r_1, r_2, \ldots, r_k\}$ for k trajectories. Then we calculate $\{d_1, d_2, \ldots, d_k\}$, where each $d_i = |r_i - t|$ and $t$ is the target integer. We count $c = \mathbb{I}_{d_i < d_{i-1}}$ and the ratio $c/(k-1)$ signifies how frequently the gap to the target decreased. If this ratio is greater than 0.6, a reward of 0.1 is obtained. Note that we only do this procedure if $k > 2$, since otherwise there are no gaps to compare with each other.

## 4 Experiments

### 4.1 Countdown Dataset

Emergent CoT properties can be observed through training LLMs on reasoning tasks. A very simple toy dataset used for this purpose is the countdown dataset [13]. It contains 490,000 pairs $(s, t)$ where $s$ is a set of $n$ two-digit positive integers and $t$ is the target integer. The goal is to achieve the target by using arithmetic operations on the numbers in $s$ not more than once. For example, a sample pair will be $s = (95, 11, 56)$ and $t = 28$. Then the desired solution is $95 - (11 + 56)$. Since arithmetic operations can be commutative and/or distributive, multiple permutations of the solution may exist such as $95 - (56 + 11)$ or $95 - 56 - 11$. In our experiments, $n = \{3, 4\}$. The prompt also contains the instruction that the reasoning should be contained within `<think>` and `</think>` tags, while the answer in `<answer>` and `</answer>` tags.

| Model | Variant | Entropy | Dr. GRPO | Ref Model (KL) | Success Rate |
|---|---|---|---|---|---|
| Qwen-2.5 3B | Pre-trained | n/a | n/a | n/a | 11.03% |
| | GRPO | ✗ | ✗ | ✓ | 70.26% |
| | GRPO | ✓ | ✗ | ✓ | 73.44% |
| | Dr. GRPO | ✓ | ✓ | ✓ | 77.63% |
| | Dr. GRPO | ✓ | ✓ | ✗ | 72.51% |
| Qwen-2.5 7B | Pre-trained | n/a | n/a | n/a | 28.80% |
| | GRPO | ✓ | ✗ | ✓ | 81.83% |
| | Dr. GRPO | ✓ | ✓ | ✓ | 83.30% |
| | Dr. GRPO | ✓ | ✓ | ✗ | 80.18% |

Table 1: Ablation study on the objective function in GRPO and Dr. GRPO while training Qwen-2.5 models on the Countdown dataset. ✓ represents that feature has been included in the training phase, while ✗ means the feature is excluded. n/a stands for "not applicable".

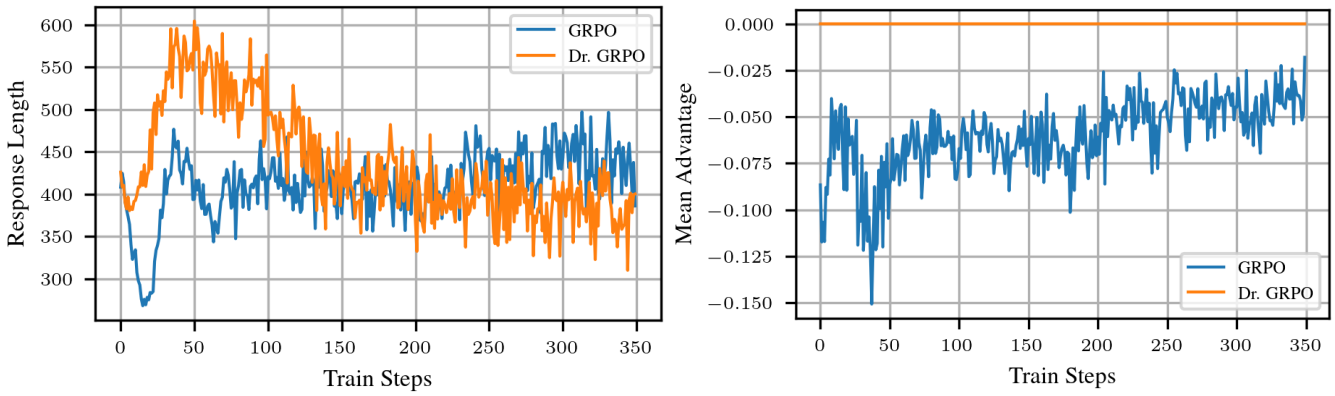
Finding these combinations is not trivial. Given 4 integers and 4 arithmetic operations, a simple combinatorial calculation says $4! \times 4^3 = 1536$ different combinations are possible. With simple logic and reasoning, most of these combinations can be discarded since they will give rise to either fractions or negative or very large numbers. Thus, this simple dataset helps to gain insight into emergent CoT and self-reflection capabilities on an LLM trained using GRPO.

### 4.2 Training Details

All experiments were carried out on the pre-trained Qwen-2.5 3B and 7B models from the Hugging face repository. The countdown dataset consists of 327680 train samples and 1319 test samples. Multiple experiments were conducted with various settings as shown in Tables 1 and 2. LLM training is implemented through the verl library (ver: 0.1) [18]. The experiments were performed with mixed-precision on 2x Nvidia H100 GPUs. The models were fine-tuned with a train batch size of 128, using AdamW with a learning rate of $10^{-6}$. When trained with a reference model, the KL coefficient was kept at $10^{-3}$. The entropy coefficient was also kept at $10^{-3}$. The max response length during generation was 1024 and the number of rollouts per prompt was kept at 5 with a temperature of 1.0.

### 4.3 Results & Discussion

**Research Question 1: How successful is GRPO in ensuring that the model consistently arrives at correct solutions?** Table 1 compares the Qwen-2.5 3B and 7B models and their multiple variants. It can be observed that GRPO is able to achieve much higher success rates than the original pre-trained models. In the 3B case, GRPO training achieves 70.26%, while initially only 11.03% success rate. Figures 1a and 1b show how the mean response length and the mean advantage change during GRPO training.




(a) Mean generation length over training steps

(b) Mean advantage over training steps

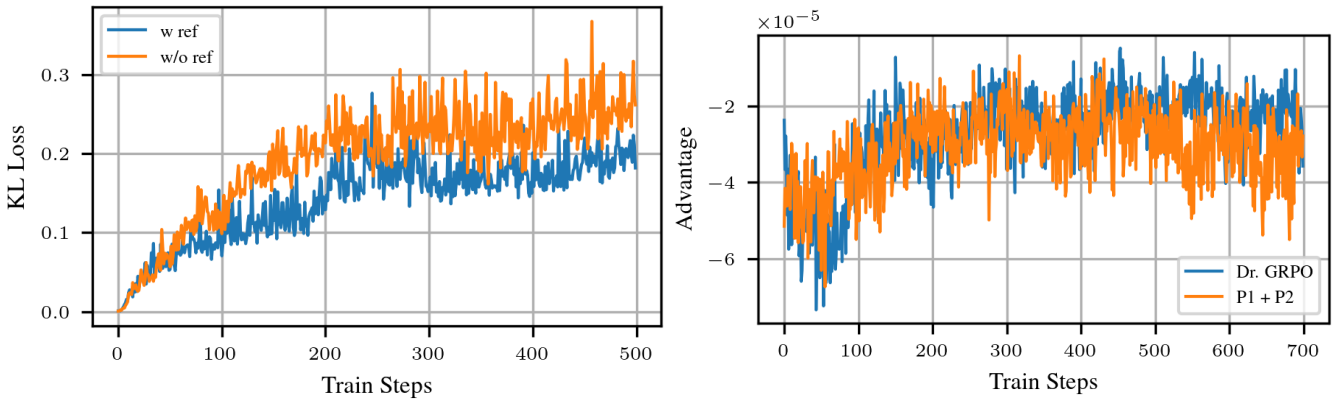
Figure 1: Comparison between GRPO and Dr. GRPO while training a Qwen-2.5 3B model

An increase in response length initially indicates that the model tries to learn reasoning for the given problem. On the other hand, the absolute mean advantage is higher due to model exploration initially and generation of diverse responses. Similar observations are also made for the Qwen-2.5 7B model.

**Research Question 2: How does entropy regularization influence model training with GRPO?** The entropy term in the objective function plays a crucial role in model training. As shown in Table 1, the variant with the entropy term achieves a higher success rate (73.44%) than the GRPO-trained model (70.26%). It seems that by encouraging higher entropy, the model maintains greater uncertainty, which fosters exploration and allows it to discover diverse reasoning strategies that might lead to the correct answer. We observe that a coefficient of $\lambda = 0.001$ is best during training. Higher coefficients can inject too much randomness and make the model underperform.

**Research Question 3: How does Dr. GRPO compare to GRPO?** As observed in Table 1, Dr. GRPO outperforms GRPO based on the success rate of the trained policy on the success rate. We observe a significant 3.8% improvement in the success rate of Dr. GRPO over GRPO. Figures 1a and 1b compare the mean response length and the mean advantage, respectively, during the training progress. Since Dr. GRPO drops the $\frac{1}{|o_i|}$ term, therefore there is a lesser drop in the response length initially. Further increase and then decrease shows that the model learns to predict the correct trajectories or combinations faster. Similarly, due to removal of the standard deviation term from the advantage calculation, there are few large negative advantages. This is reflected in Fig. 1b where the mean advantage when training with Dr. GRPO is almost 0. However, it is not exactly 0 as can be seen in Fig. 2b. Hence, Dr. GRPO significantly improves the reasoning capabilities of the trained policy.

**Research Question 4: Is the reference model a crucial component of GRPO, or can a comparable performance be achieved in its absence?** As mentioned above, [9] drops the reference model and claims that it should not affect the model as rule-based verifiers are being used. A comparison has been made between keeping the KL-term in the objective function and dropping it. There is a significant difference in the model performance as seen in Table 1. A performance drop from 77.63% to 72.51% is observed for the 3B model. Therefore, the KL term plays an important role in training. Not only does it keep the model similar to the original model, but it also maintains good language generation capabilities of the model. Fig. 2a shows the comparative difference between training with and without the reference




(a) KL loss over training steps

(b) Mean generation length over training steps

Figure 2: Comparison between Dr. GRPO with and without the KL term and mean generation length for Dr. GRPO base policy and with additional P1 + P2 prompts.

policy respectively. Large KL distances often imply poor normal generation of sentences and forgetting learned data.

**Research Question 5: Does a larger LM improve success rates?** Empirical studies have shown that loss metrics such as perplexity decrease in a predictable, approximately power-law fashion when factors such as model or dataset size are scaled up [19, 20]. Therefore, an improvement in the success rates is expected when comparing the Qwen-2.5 7B model with the 3B model. We observe that there is indeed an increase from 73.44% to 81.83% as indicated in Table 1. Even the pre-trained 7B model has a much higher success rate (28.80%) than the 3B model (11.03%). We also observed similar improvements in the Dr. GRPO algorithm while training the 7B model. This improvement can be attributed to the larger model's more robust reasoning and exploration capacity.

**Research Question 6: Does additional prompt engineering or reward shaping aid training progress?** Table 2 shows how the additional prompts (P1 and P2) and reward signals (R1 and R2) affect training of the 3B and 7B models.

Using a generic prompt such as P1 improves the success rate by encouraging more logical and deliberate reasoning steps. Adding P2, which enforces the use of multiple `<think>` tags, further boosts performance. However, this effect appears to be highly task-specific and may not be generalized to other math-related problems. In particular, these improvements are observed consistently only for the 7B model; in contrast, the 3B model shows a significant drop in success rate when similar prompting strategies are applied. Log analysis indicates that during training, the smaller model ignores prompt instructions. The additional capacity of the 7B model probably enables it to better interpret and adhere to complex prompt constraints.

Reward shaping was found to be inconsistent and often ineffective in improving model performance beyond what was achieved with the base algorithm, as seen in Table 2. In practice, the additional reward signals did not consistently translate into higher success rates. This limitation occurs because the model misinterprets the additional rewards, sometimes exploiting them in unintended ways. For example, during training with the R1 reward, we observed that the model began to collapse all its reasoning steps into a single line rather than using multiple `<think>` tags as intended, probably to maximize the reward with minimal structural

| Model (Variant) | P1 | P2 | R1 | R2 | Success Rate |
|---|---|---|---|---|---|
| Qwen-2.5 3B (Dr. GRPO) | ✗ | ✗ | ✗ | ✗ | 77.63% |
| | ✓ | ✗ | ✗ | ✗ | 74.07% |
| | ✓ | ✓ | ✗ | ✗ | 71.32% |
| | ✓ | ✓ | ✓ | ✗ | 71.77% |
| | ✓ | ✓ | ✗ | ✓ | 71.66% |
| Qwen-2.5 7B (Dr. GRPO) | ✗ | ✗ | ✗ | ✗ | 83.30% |
| | ✓ | ✗ | ✗ | ✗ | 84.57% |
| | ✓ | ✓ | ✗ | ✗ | 87.70% |
| | ✓ | ✓ | ✓ | ✗ | 86.04% |
| | ✓ | ✓ | ✗ | ✓ | 84.38% |

Table 2: Qwen-2.5 models trained with Dr. GRPO with additional prompts and/or reward signals.

effort. These observations suggest that designing effective reward shaping strategies for complex reasoning tasks is non-trivial and may inadvertently encourage undesirable behaviors.

A sample output with numbers 20, 19, 14 and 24, and target 30 is given below. Here, one can observe that the number 19 is not required in the final answer. We already provided the model with the prompt "Let me solve this step by step" before the following output.

> <think>Starting with the numbers 20, 19, 14, 24, I need to find a way to get 30. Maybe I can add 20 and 14 to get 34, then subtract 19 to get 15, and then add 24 to get 39. Not working.</think>
> <think>20 + 19 = 39, 39 - 14 = 25, 25 + 24 = 49. Nope.</think>
> <think>20 + 19 = 39, 39 - 24 = 15, 15 + 14 = 29. Still not 30.</think>
> <think>24 - 19 = 5, 5 + 14 = 19, 19 + 20 = 39. Nope.</think>
> <think>24 - 14 = 10, 10 + 20 = 30. Bingo!</think>
> <answer> (24 - 14) + 20 </answer>

**Research Question 7: Do these results generalize to similar datasets?** We tested the final trained best-performing model (P1 + P2 7B) with two datasets: a similar toy dataset and the GSM8K dataset [21]. Although the model was trained with the prompts P1 + P2, we test if having P1 + P2 in the test samples improves the success rate.

The second toy dataset, referred to as the hidden operations challenge, consists of samples in which the arithmetic operators of an arithmetic equation are hidden. For example, a sample input might be "2 _ 3 _ 5 = 11," where the model must correctly insert '*' and '+' to satisfy the equation. On this dataset, the fine-tuned model achieves a success rate of 80.56% when the P1 + P2 prompts are provided at test time, compared to 75.42% without these prompts. This indicates that the structured prompt helps the model

perform multiple guesses more systematically and manage each intermediate step within explicit `<think>` tags. In particular, the original pretrained Qwen-2.5 7B reference model achieves only 11.11% accuracy under the same conditions.

We further tested the model on the GSM8K dataset, which contains 8.5K grade-school-level math word problems. We perform inference on about 1.3K test samples, since the pre-trained model was already trained with the train data [15]. Since the pretrained model was already exposed to the training split during its original development [15], we evaluated on approximately 1.3K held-out test samples. The original Qwen-2.5 7B model achieves a reported accuracy of 85.4% on this benchmark. Our fine-tuned model reaches 75.61% accuracy when using only the standard dataset prompt. However, when we supplement the initial prompt with the prompt format, the accuracy increases to 83.25%. Also, when further supplemented with P1, we observe that the model achieves 84.60% accuracy, which is within 1% of the reported accuracy. This indicates that the model not only does not differ much from the reference model but also improves when provided with appropriate prompts. In contrast, the addition of P2 significantly reduces performance to around 63%, as it adds unnecessary complexity to the output.

## 5 Conclusion

In this work, we systematically analyzed how GRPO and its improved variant, Dr. GRPO, affect the emergence of chain-of-thought (CoT) reasoning in large language models. By combining weak verifiers, prompt engineering, and carefully designed reward schemes, we investigated how each component influences reasoning performance on the Countdown dataset and beyond.

Our results show that GRPO is successful in training a model to learn reasoning tasks on which it was not explicitly trained previously. Dr. GRPO further mitigates known biases, improving success rates. Entropy regularization was found to encourage exploration and maintain reasoning diversity without sacrificing correctness. The KL term also proved to be important while training the model. We also demonstrated that prompt engineering, especially the use of structured `<think>` tags, meaningfully improves reasoning quality in larger models. Reward shaping, while theoretically promising, was inconsistent in practice and at times encouraged unintended shortcut behaviors, highlighting the need for careful design of auxiliary rewards. Finally, we showed that the improvements generalize to similar reasoning tasks and show competitive performance on standard math benchmarks like GSM8K when supported by appropriate prompts.

Future work can focus on extending this work to more diverse open-domain reasoning tasks and real-world problem settings. Designing robust verifiers or adaptive reward mechanisms may be effective in improving success rates. Self-play and curriculum learning can help the model progressively solve more complex problems or larger numerical ranges by generating and solving increasingly challenging examples autonomously. With self-play, the model expands its reasoning capabilities without relying solely on static datasets. Finally, it will be valuable to investigate how these methods generalize to larger LLMs and to confirm whether the trends observed here hold at larger scales.

## References


[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. (cited on pages 1, 2, and 5)

[2] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022. (cited on pages 1 and 2)

[3] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024. (cited on pages 1 and 2)

[4] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. (cited on page 1)

[5] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. (cited on page 1)

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. (cited on pages 1, 2, and 3)

[7] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. (cited on pages 1 and 2)

[8] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. (cited on pages 1, 3, and 6)

[9] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025. (cited on pages 1, 2, 4, and 8)

[10] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999. (cited on page 2)

[11] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017. (cited on page 2)

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. (cited on page 2)

[13] Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. https://github.com/Jiayi-Pan/TinyZero, 2025. Accessed: 2025-01-24. (cited on pages 2 and 6)

[14] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022. (cited on page 3)

[15] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. (cited on pages 4 and 11)

[16] Quy-Anh Dang and Chris Ngo. Reinforcement learning for reasoning in small llms: What works and what doesn't. *arXiv preprint arXiv:2503.16219*, 2025. (cited on page 4)

[17] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024. (cited on page 5)

[18] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024. (cited on page 7)

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. (cited on page 9)

[20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. (cited on page 9)

[21] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. (cited on page 10)