

A BASELINE IMPLEMENTATION DETAILS

In this section, we provide background on reinforcement learning (RL) for large language models (LLMs) and describe the objective that different baselines optimize.

A.1 RL FOR LLMs

Let $\pi_\theta(a \mid s)$ denote a policy parameterized by θ , which defines the probability of taking action a in state s . A trajectory of length T is denoted by $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$, and the total reward of a trajectory is $R(\tau)$. In the context of language models, the initial state s_0 corresponds to the input question, actions correspond to generated tokens, and subsequent states represent the question along with the partial sequence of answer tokens.

RL training using Reinforce (Williams, 1992) optimizes the following objective:

$$\theta^* = \operatorname{argmax}_\theta \mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{y \sim \pi_\theta(\cdot \mid x)} R(x, y) \quad (1)$$

Here, x is a question sampled from the distribution of questions \mathcal{D} , y is a response sampled from the LLM $\pi_\theta(\cdot \mid x)$, and $R(x, y)$ is the reward for generating response y to question x . In our graph search setting, we prompt the LLM to output its final answer inside `\boxed{ }`, so the reward function simply extracts the string in `\boxed{ }` and compares it to the true path.

A.2 DR. GRPO (Liu et al., 2025)

The objective in Equation 1 typically exhibits high variance. To reduce this variance, recent methods such as GRPO (Shao et al., 2024) Dr. GRPO (Liu et al., 2025) subtract a baseline from the reward, which is computed by generating multiple rollouts for a given question and taking the average reward. In our setting, we perform on-policy training; thus, the importance ratio and advantage clipping terms in the objective from equation 3 in (Liu et al., 2025) vanish, resulting in the following final objective:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{x \sim \mathcal{D}, \{y_i\}_{i=1}^G \sim \pi_\theta(\cdot \mid x)} \frac{1}{G} \sum_{i=1}^G \sum_{t=1}^{|y_i|} \hat{A}_{i,t} \nabla_\theta \log \pi_\theta(y_{i,t} \mid y_{i,<t}) - \beta \mathbb{D}_{\text{KL}}(\pi_\theta \parallel \pi_{\text{ref}}) \quad (2)$$

where $\hat{A}_{i,t}$ is computed as $R(x, y_i) - \frac{1}{G} \sum_{j=1}^G R(x, y_j)$. We set $G = 5$ for all our experiments.

Intuitively, the objective increases the likelihood of tokens that lead to positive outcomes and decreases the likelihood of tokens that do not. Note that this loss formulation weights all steps in the reasoning chain equally, i.e., $\hat{A}_{i,t}$ is independent of t .

A.3 VINEPPO (Kazemnejad et al., 2024)

Methods like GRPO and Dr. GRPO weigh all tokens in a reasoning chain equally. However, this may not be optimal, as not all steps in a reasoning chain contribute equally to solving a problem. For example, identifying lemmas to use in proving a theorem may be more important than generating tokens that are included merely for grammatical correctness. VinePPO addresses this problem by performing credit assignment to weigh more important steps higher and less important steps lower. It does so by computing step level advantages instead of trajectory level advantages.

Given a question $x \sim \mathcal{D}$, and a reasoning chain $y \sim \pi_\theta(\cdot \mid x)$, VinePPO divides the reasoning chain y into steps or chunks $y_{c_1}, y_{c_2} \dots y_{c_n}$, and finally the gradient of the objective is computed as

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{i=1}^n \hat{A}_{y_{c_i}}^{\pi_\theta} \sum_{t=s_{c_i}}^{e_{c_i}} \nabla_\theta \log \pi_\theta(y_t \mid y_{<t}, x) \quad (3)$$

where $\hat{A}_{y_{c_i}}^{\pi_\theta}$ denotes the advantage of step y_{c_i} , with s_{c_i} and e_{c_i} representing the start and end indices of the i^{th} chunk, respectively. The advantage at each step is estimated using Monte Carlo rollouts from the state before and after that step. In other words, given a trajectory $y = y_{c_1} \oplus y_{c_2} \oplus \dots \oplus y_{c_n}$ where \oplus denotes the concatenation operation $\hat{A}_{y_{c_i}}^{\pi_\theta}$ is computed as follows:

$$\hat{A}_{y_{c_i}}^{\pi_\theta} = V^{\pi_\theta}(\oplus_{j=1}^{j=i} y_j) - V^{\pi_\theta}(\oplus_{j=1}^{j=i-1} y_j) \quad (4)$$

where $V^{\pi_\theta}(s)$ is computed by taking the average reward recieved from K Monte Carlo rollouts. In our experiments we set $K = 3$. Note that we omit the KL penalty term from Equation 3 for the sake of simplicity. For VinePPO, we use the code released by the authors [here](#).

A.4 PROGRESS REWARDS (SETLUR ET AL., 2025A)

Like the VinePPO objective in Equation 3 progress rewards also aim to compute step-level advantages. However, their objective differs from VinePPO in two key ways: (i) they estimate the advantages in Equation 4 under a policy different from the one being optimized, which they refer to as the prover policy, and (ii) combine the step-level advantages with the trajectory-level reward to form the final objective:

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{i=1}^n (R(y) + \alpha \cdot \hat{A}_{y_{c_i}}^\mu) \sum_{t=s_{c_i}}^{e_{c_i}} \nabla_\theta \log \pi_\theta(y_t | y_{<t}, x) \quad (5)$$

where μ is `Best-of-4`(π_{ref}). Similar to equation 4 they compute step advantages under the prover policy μ as:

$$\hat{A}_{y_{c_i}}^\mu = V^\mu(\oplus_{j=1}^{j=i} y_j) - V^\mu(\oplus_{j=1}^{j=i-1} y_j) \quad (6)$$

In their work, a neural network is trained to approximate $V^{\pi_{\text{ref}}}(s)$ by collecting a dataset of reasoning trajectories sampled from π_{ref} . For simplicity, we instead rely on rollouts to compute $V^{\pi_{\text{ref}}}(s)$. With this, we optimize the following objective:

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{i=1}^n (\hat{A}(y) + \alpha \cdot \hat{A}^\mu(y_{c_i})) \sum_{t=s_{c_i}}^{e_{c_i}} \nabla_\theta \log \pi_\theta(y_t | y_{<t}, x) \quad (7)$$

Notice the change from Equation 5 where we replace $R(y)$ with $\hat{A}(y)$. We make this substitution because we operate in a group-style setting with access to multiple rollouts for a question. Using $\hat{A}(y)$ instead of $R(y)$ yields a lower-variance estimate of the gradient. $\hat{A}(y)$ is computed as ($R(y)$ – average reward over G rollouts).

A.5 BEST-OF-N FINETUNING (CHOW ET AL., 2025)

[Chow et al. \(2025\)](#) introduce inference-aware fine-tuning that explicitly optimizes for BoN performance. The hope is optimizing for BoN performance could result in the model sampling *diverse* responses. A key result is Lemma 3, which provides a BoN-aware policy gradient estimator under binary rewards. By introducing asymmetric weighting functions that depend on the probability of failure, the method upweights correct predictions on hard inputs while redistributing mass away from unreliable outputs.

Assuming the reward $R(x, y) \in \{0, 1\}$. Then the gradient of the BoN-aware RL objective (Equation 6) with respect to the model parameters θ is

$$\mathbb{E}_{x \sim D} [\mathbb{E}_{y \sim \pi_{\text{bon}, R=1}} [\nabla_\theta \log \pi_\theta(y | x)] g_N^+(P_{\text{fail}}(x)) - \mathbb{E}_{y \sim \pi_{\text{bon}, R=0}} [\nabla_\theta \log \pi_\theta(y | x)] g_N^-(P_{\text{fail}}(x))], \quad (8)$$

where the sample-dependent weights are

$$g_N^+(p) = \frac{N p^{N-1}}{1 - p^N}, \quad g_N^-(p) = \frac{N (1 - p^{N-1})}{1 - p^N}.$$

B HYPERPARAMETERS

Table 1 lists the hyperparameters that are common across all methods. Since we do on-policy training parameters such as the clipping ratio (ϵ) and `ppo-epochs` are not applicable.

Hyperparameter	Value
Learning Rate	1×10^{-6}
Effective Batch Size	32
Number of rollouts per data point	5
Max Prompt Length	1024
Max Response Length	4096
KL Coefficient (β)	1×10^{-3}
Temperature	0.6
Top p	0.999
Micro Batch Size	4
Discount Factor (γ)	1
Base Model	Qwen2.5/Qwen-1.5B-Instruct

Table 1: Common hyper-parameters used in our experiments.

B.1 VINEPPO

We estimate the value of an intermediate state, $V^{\pi_\theta}(s)$, using three rollouts from the current policy.

B.2 REWARDING PROGRESS

We estimate the value of an intermediate state, $V^{\pi_{\text{ref}}}(s)$, using three rollouts from the reference policy. The parameter α in Equation 7 is set to 5, following the recommendation in Setlur et al. (2025a).

B.3 BEST-OF-N AWARE FINETUNING

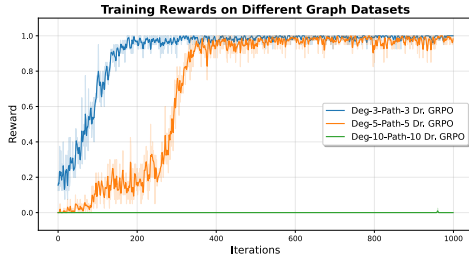
For N used in Best-of-N, we use $N = 8$. We clip the sample dependent weights to $[-3, 3]$. We decay the KL coefficient from 0.1 to 0.001.

C PROMPTS

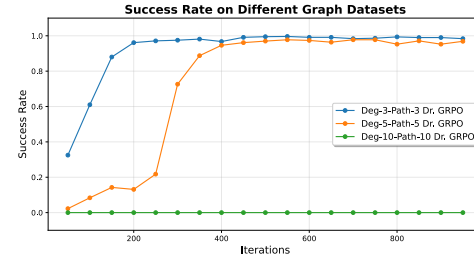
For our experiments, we prompt the Qwen2.5/Qwen-1.5B-Instruct model in the following manner

Prompt for path finding problem
<p>Given a bi-directional graph in the form of space separated edges, output a path from source node to the destination node in the form of comma separated integers.</p> <p>For this question the graph is 81,252 97,124 285,182 97,285 97,81 124,199</p> <p>The source node is 97</p> <p>The destination node is 252</p> <p>Please reason step by step, and put your final answer within <code>\boxed{}</code>.</p>

During training, the string inside `\boxed{}` is extracted and compared against the ground truth path to assign the reward.



(a) Rewards obtained during training Dr. GRPO on the graph search task with varying levels of difficulty.



(b) Success Rate of Dr. GRPO on held out test sets of different levels of difficulty. The model manages to solve simpler variants ([Degree-3-Path-3](#) and [Degree-5-Path-5](#)), but is unable to solve the harder [Degree-10-Path-10](#) variant.

Figure 7: Dr. GRPO is able to solve easier instance of the graph search task.

D ADDITIONAL RESULTS

Easier versions of the task are solvable: To rule out any implementation issues and verify that the graph search task is not inherently unsolvable, we experiment with simpler variants of the task, such as the Degree-3-Path-3 and Degree-5-Path-5 graphs. As shown in Figs. [7a](#) and [7b](#) RL training using outcome rewards is effective on these simpler variants, but remains ineffective on the harder Degree-10-Path-10 dataset.