KNAPSACK RL: UNLOCKING EXPLORATION OF LLMS VIA OPTIMIZING BUDGET ALLOCATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) can self-improve through reinforcement learning, where they generate trajectories to explore and discover better solutions. However, this exploration process is computationally expensive, often forcing current methods to assign limited exploration budgets to each task. This uniform allocation creates problematic edge cases: easy tasks consistently succeed while difficult tasks consistently fail, both producing zero gradients during training updates for the widely used Group Relative Policy Optimization (GRPO). We address this problem from the lens of exploration budget allocation. Viewing each task's exploration as an "item" with a distinct "value" and "cost", we establish a connection to the classical knapsack problem. From this, we derive an optimal assignment rule that transfers exploration budgets from easy tasks to challenging ones. When applied to GRPO, our method increases the effective ratio of non-zero policy gradients by 20–40% during training. As a computational "free lunch", it also enables substantially larger exploration budgets (e.g., 93 rollouts) for especially challenging tasks—budgets that would be computationally prohibitive under uniform allocation. These improvements translate to meaningful gains on mathematical reasoning benchmarks, with average improvements of 2–4 points and peak gains of 9 points on specific tasks. Notably, achieving comparable performance with traditional homogeneous allocation would require about 2x the computational resources.

1 INTRODUCTION

The remarkable capabilities of Large Language Models (LLMs) have led to their widespread application across various domains (OpenAI, 2025; Comanici et al., 2025; Anthropic, 2025; Meta, 2025; Yang et al., 2025). While pre-training on vast text corpora endows LLMs with general knowledge and linguistic fluency, fine-tuning them for specialized tasks often necessitates more targeted optimization beyond pre-training. Reinforcement Learning (RL) has emerged as a powerful paradigm for this purpose (Ouyang et al., 2022; Li et al., 2024; Guo et al., 2025), enabling LLMs to iteratively self-improve by interacting with environments. A popular instantiation is RL with verifiable rewards (Lambert et al., 2024), where LLMs generate responses and receive binary (true/false) feedback based on their outcomes, iteratively refining their internal policies to search for optimal solutions. Initially pioneered in mathematical reasoning (Jaech et al., 2024), this framework has since been extended to domains like coding (Luo et al., 2025a) and agentic tasks (Team et al., 2025).

A core challenge in these applications is *exploration*—sampling diverse trajectories to find better solutions. This process is computationally expensive in practice due to sequential nature of autoregressive generation. As such, most RL pipelines use a small number of rollouts per prompt (e.g., 8) for exploration. However, this uniform allocation strategy could lead to some problematic outcomes. For example, in the Group Relative Policy Optimization (GRPO) (Shao et al., 2024) algorithm, meaningful learning signals (gradients) only emerge when both successful and failed attempts are present in the same batch. With a uniform budget, easy tasks often result in all-success outcomes, and hard tasks in all-failure outcomes, leading to near-zero gradients and stalled learning. This issue has been well-documented in previous research (Yu et al., 2025; Chen et al., 2025a), and we approach it from the broader perspective of strategic exploration budget allocation.

We argue the fundamental problem is the mismatch between a task's difficulty and its assigned exploration budget. Hard tasks, which require extensive (could even require more than 100) exploration to find useful trajectories, receive too little effort under a uniform rule. Easy tasks, which require

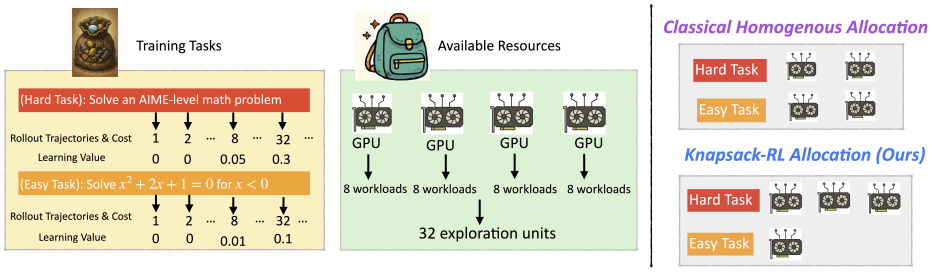


Figure 1: Illustration of our framework for allocating exploration budgets among tasks from computational resources. We model each task as an item with learning value and computational cost, then solve the allocation problem using Knapsack optimization.

minimal exploration, waste compute by being over-sampled. Thus, a heterogeneous and customized exploration allocation strategy is preferred.

To this end, we introduce a knapsack-based formulation: each task, when assigned a certain budget, can be conceptualized as an 'item" with an associated value (learning potential) and cost (computational effort of exploration). The allocation problem is thus equivalent to the classical knapsack problem (Mathews, 1896; Pisinger & Toth, 1998), where the objective is to maximize total value under a fixed global budget. We refer to this approach as Knapsack RL; see Figure 1 for illustration. When applied to the popular GRPO framework, our method enables a dynamic, heterogeneous allocation of exploration budgets, which allows sufficient exploration on training tasks.

Empirically, across Qwen series models (Yang et al., 2024; 2025) sized from 1B to 7B, we observe a 20-40% improvement in effective gradient ratios, translating into more reliable policy improvements and average performance gains of about 2-4 points on several challenging benchmarks. To get a better sense of this improvement, we note that achieving comparable improvements with uniform allocation would require nearly 2x the computation. We present this as a proof-of-concept, demonstrating a promising direction to boost the effectiveness of RL.

2 Preliminary

Following (Ouyang et al., 2022; Shao et al., 2024), we model language generation as autoregressive sampling from a conditional probability distribution $\pi_\theta(y|x)$, where x represents the input prompt and y represents the generated response. The parameter θ denotes the trainable parameters. Our goal is to improve the language model via RL by maximizing the expected performance of responses generated from the model distribution π_θ:

$$\max_\theta \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[r(x,y)] \tag{1}$$

In this paper, we focus on RL with verifiable rewards. Specifically, let $y = (\texttt{CoT}, \texttt{answer})$ denote the concatenation of Chain-of-Thought (CoT) (Wei et al., 2022) reasoning steps $\texttt{CoT}$ and the final solution $\texttt{answer}$. The reward function $r(x, y)$ is defined as:

$$r(x,y) = \mathbb{I}(\texttt{answer} \text{ is correct with respect to } x), \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function and $r \in \{0, 1\}$ is binary (1 for correct, 0 for incorrect). This outcome-based reward formulation has been widely adopted (see e.g., (Guo et al., 2025) and references therein) and has been shown to effectively incentivize reasoning abilities (Wen et al., 2025).

Algorithm 1 RL with Classical Homogeneous Budget Allocation

1: **for** iteration $t = 1, 2, \ldots$ **do**
2: Sample a mini-batch of prompts $(x_1, \ldots, x_M)$
3: Generate N responses for each prompt x_i ▷ *Budget Allocation*
4: Evaluate the rewards (e.g., Equation (2)) and compute the gradients (e.g., Equation (3))
5: Update model parameters with estimated gradients

To optimize Equation (1), policy gradient methods (Sutton et al., 1999) are commonly employed. Among these, REINFORCE (Williams, 1992)-style stochastic policy gradient methods have become standard since (Li et al., 2024). These methods stochastically sample N responses from π_θ and estimate gradients using direct reward feedback. Originally designed for single-task RL, this approach

is typically extended to multi-task RL by employing homogeneous exploration budget allocation. Algorithm 1 summarizes this classical framework.

In Algorithm 1, the sampling process in Line 3 corresponds to exploration in RL, where the model generates responses to search for optimal solutions. Line 5 corresponds to exploitation, updating the model to leverage feedback from data. We adopt the widely used gradient estimator from Group Relative Policy Optimization (GRPO) (Shao et al., 2024):

$$g(\theta) = \sum_{i=1}^{M} \sum_{j=1}^{N} \nabla_\theta \log \pi_\theta(y_{ij}|x_i) \cdot (r(x_i, y_{ij}) - b_i) \cdot c_i \tag{3}$$

where y_{ij} denotes the j-th sampled response for prompt x_i, and $\nabla_\theta \log \pi_\theta(y_{ij}|x_i)$ represents the gradient of the log-probability with respect to model parameters θ. The baseline b_i and normalization factor c_i are defined as: $b_i = 1/N \cdot \sum_{j=1}^{N} r(x_i, y_{ij})$ and $c_i = 1/(\sigma_i + \epsilon)$ with $\sigma_i = \sqrt{1/N \cdot \sum_{j=1}^{N} (r(x_i, y_{ij}) - b_i)^2}$ is the standard deviation of rewards for prompt x_i, and ϵ is a small constant (10^{-6}) preventing division by zero when $\sigma_i = 0$. Technically, GRPO computes relative advantages within each response group (prompt), increasing likelihood of positive responses and decreasing likelihood of negative ones.

3 Diagnosing Exploration in Homogeneous Budget Allocation

In this section, we discuss the limitations of homogeneous budget allocation for GRPO and present empirical observations that motivate our work.

3.1 Motivation

Exploration in RL is computationally expensive due to the sequential nature of autoregressive generation, often requiring substantial GPU memory and hours of computation, especially for reasoning tasks. Thus, it is critical to assess how much each collected sample actually contributes to gradient updates. For GRPO, we make the following observation.

Observation 1. *Let $g_i = \sum_{j=1}^{N} \nabla_\theta \log \pi_\theta(y_{ij}|x_i) \cdot (r(x_i, y_{ij}) - b_i) \cdot c_i$ be the gradient for prompt i. If $\sigma_i = 0$, meaning that all N sampled responses for x_i yield identical rewards (all correct or all incorrect), then $(r(x_i, y_{ij}) - b_i) = 0$ for every sample, leading to $g_i = 0$. In this case, the model receives no learning signal from that prompt.*

This phenomenon is widely recognized as a major bottleneck for GRPO in practice (Yu et al., 2025; Chen et al., 2025a). To formally track it, we introduce the metric `effective-gradient-ratio`, which measures the proportion of individual samples that contribute non-zero gradients:

$$\texttt{effective-gradient-ratio} = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} \mathbb{I}(g_{i,j} \neq 0), \tag{4}$$

where $g_{i,j} = \nabla_\theta \log \pi_\theta(y_{ij}|x_i) \cdot (r(x_i, y_{ij}) - b_i) \cdot c_i$ is the gradient contribution from the j-th sample of the i-th prompt. A higher value indicates that a larger fraction of samples are contributing useful learning signals. We also define two complementary metrics: `zero-gradient-ratio (by all positive rewards)`: proportion of prompts yielding zero gradients due to uniformly positive rewards; and `zero-gradient-ratio (by all negative rewards)`: proportion of prompts yielding zero gradients due to uniformly negative rewards.

We visualize these dynamics in Figure 2 for the `Qwen2.5-Math-7B` model trained on the `DAPO-MATH-17K` dataset. Each mini-batch contains $M = 256$ prompts with $N = 8$ rollouts per prompt. The results reveal several concerning patterns:

Low Overall Effectiveness: The effective gradient ratio consistently remains below 60%, meaning that over 40% of sampled data fails to contribute to model updates—a significant waste of computational resources.

Dynamic Training Phases: The gradient dynamics exhibit three distinct phases:

- **Early Training (0-70 iterations, approximately the first epoch):** The model struggles with most tasks, leading to predominantly all-negative rewards (green line peaks near 95%). This results in minimal learning signals being generated.

- **Mid Training (70-600 iterations):** As the model improves, it begins solving some tasks while still failing others, creating the mixed outcomes necessary for effective gradients. The effective gradient ratio can maintain above 40% during this phase.
- **Late Training (600+ iterations):** Tasks become increasingly easy, leading to a rise in all-positive rewards (orange line increases to 40%). Simultaneously, challenging tasks still result in all-negative rewards (the green line fluctuates around 20%). As a result, the effective-gradient-ratio steadily decreases to about 20% by 1000 iterations.

We provide theoretical analysis toward understanding the above empirical observations in the next section.

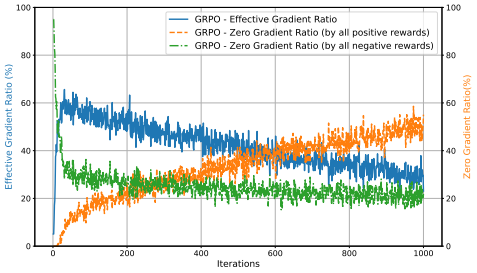


Figure 2: The ratio of effective gradients and zero gradients during training.

3.2 Theoretical Analysis

We model reward outcomes as Bernoulli random variables to analyze the exploration budget required.

Definition 1 (Success rate)**.** *We define the success rate p on a prompt x as the probability that the model generates a correct response: $p_i \equiv p(x_i) = \mathbb{E}_{y\sim\pi_\theta(\cdot|x_i)}[r(y|x_i)] = \Pr[r(y|x_i) = 1]$.*

This formulation allows statistical analysis of stochastic gradients. For N sampled responses $y_{i1}, \ldots, y_{iN}$ on a prompt x_i, the probability that both correct and incorrect samples are observed is:

$$\begin{aligned}\mathbb{P}(g_i \neq 0) &= 1 - \mathbb{P}[\text{all rewards are the same}] = 1 - \mathbb{P}[\text{all rewards are 1's}] - \mathbb{P}[\text{all rewards are 0's}] \\ &= 1 - p_i^N - (1-p_i)^N.\end{aligned}$$

This raises the question: how large must the sampling budget N be to obtain a non-zero gradient? We answer this from two perspectives: high-probability guarantees and expected sample complexity.

Theorem 1 (Exploration Budget)**.** *Given a prompt with the success rate $p \in (0, 1)$, we have that*

- ***High probability bound:*** *For any $\alpha \in (0, 1)$, to ensure $\mathbb{P}(g_i \neq 0) \geq \alpha$, it suffices to take $N \gtrsim \frac{\ln(1-\alpha)}{\ln(\max\{p_i, 1-p_i\})}$.*
- ***Expected number of rollouts:*** *Let N^{first} denote the number of independent rollouts required until $g_i \neq 0$ is achieved for the first time. Its expectation is: $\mathbb{E}[N^{\text{first}}] = 1/p + 1/(1-p) - 1$.*

Please refer to Appendix D for the proof. To illustrate, for example, if $p = 0.5$, we need 3 samples on average to obtain a non-zero gradient. For a hard task with $p = 0.01$, we require 100 samples, and to achieve a 90% chance of non-zero gradient, we would need 229 samples.

We show the theoretical predictions in Figure 3. We employ the `Qwen2.5-Math-7B-Instruct` model to generate 256 responses for 1,000 prompts from the `DAPO-Math-17K` dataset. Then we estimate p and compute the minimal budget N needed for $g_i \neq 0$ from the data. We exclude prompts that with empirical success rate of 0.0 or 1.0, because our exploration budget 256 is not sufficient. The results show that a typical budget of $N = 8$ only covers tasks with $p \in [0.1, 0.9]$. For tasks with $p \approx 0$ or $p \approx 1$, even increasing N to 16 or 32 is insufficient. Overall, our analysis shows that the sampling budget required for meaningful gradients could be much larger than what is practically used. This also helps explain the low effective gradient ratio observed in Figure 2.

Existing practices typically address this kind of insufficient exploration challenge in two ways:

- **Increasing the exploration budget uniformly.** This involves raising N—for example, from 8 to 16 or even 32—which could help address exploration on extremely hard or easy tasks and improve the effective gradient ratio. However, setting a very large value for N, such as $N = 100$, is often impractical due to prohibitive computational costs.
- **Filtering hard and easy prompts.** Tasks that are too easy or too hard are dropped. This kind of approach is leveraged in (Team et al., 2025; Yu et al., 2025). However, as we have seen, the proportion of prompts yielding zero gradients due to all-negative rewards (the green line in Figure 2) is around 20% in late training, indicating many tasks are not yet fully solved. If we simply filter these prompts, we may close off a crucial source for RL, where meaningful learning often comes

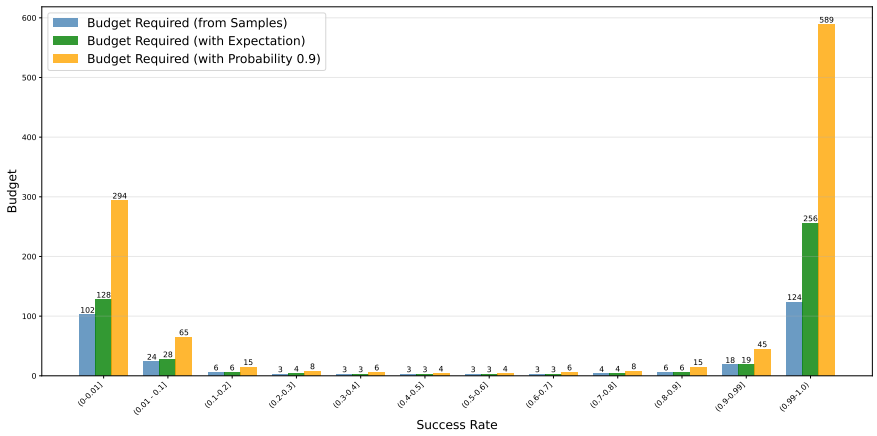


Figure 3: Exploration budget required to ensure non-zero gradients based on success rate. Note that success rates with in the same bins are grouped from real samples, which may not be symmetry, rendering the exploration budget may not be symmetry as the theory suggests.

from converting failures into successes. That is, removing hard prompts deprives the model of opportunities to practice on challenging examples, limiting the information available to LLMs.

In this work, we favor addressing this issue by scaling exploration budgets, but recognize that this first approach presents a fundamental computation-exploration dilemma. This tension motivates our pursuit of a more principled solution for allocation of exploration budgets.

4 Proposed Approach: Knapsack-based RL

In this section, we introduce our approach to address the exploration-and-computation dilemma. Crucially, our goal is not to demand additional computational resources, as these are typically fixed by the user's constraints. Given these computational resources as a fixed pool, we aim to implement a centralized allocation strategy: assigning customized exploration budgets for each task.

The central technical question is: **given a fixed total budget, what is the optimal allocation for RL exploration?** Our key insight is that task difficulty alone does not dictate the optimal allocation—tasks also differ in their *value*. Easy tasks provide limited benefit, since correcting small mistakes leads to only incremental gains, whereas solving harder tasks can yield substantial improvements. This motivates reallocating budget from easier tasks to more challenging ones. In short, effective allocation must jointly account for both *exploration cost* and *learning value*.

We formalize the above idea as a constrained optimization problem:

$$\max_{N_1,\ldots,N_M} \sum_{i=1}^{M} \mathrm{Value}(N_i, p_i) \tag{5}$$
$$\text{subject to} \sum_{i=1}^{M} N_i = N_{\text{total}}, \quad N_{\text{low}} \le N_i \le N_{\text{up}}, \quad N_i \in \mathbb{Z}^+,$$

where N_i is the number of trajectories allocated to prompt x_i, and p_i is the success rate. The bounds N_{low} (e.g., 2) and N_{up} (e.g., 128) allow to enforce coverage and prevent degenerate allocations. The total budget N_{total} is usually set to $N \times M$ to match the homogenous allocation rule.

This optimization problem exactly matches the structure of a classical knapsack problem (Pisinger & Toth, 1998). Each prompt x_i can be thought of as an item with a "weight" given by its allocated budget N_i and a "value" of $\mathrm{Value}(N_i, p_i)$. The objective is to choose budget allocations N_i that maximize the total value while keeping the overall cost within the knapsack capacity $M \times N$.

4.1 Formulation of Task Value

In this section, we substantiate the above framework with the proposed idea. We recognize that homogeneous budget allocation fails to take the task value into consideration. For GRPO, we address

this issue by defining the value of assigning N_i exploration budget units to prompt x_i as

$$\text{Value}(N_i, p_i) = \text{ProbNonZeroGradient}(N_i, p_i) \times \text{InfoGain}(p_i),$$

where $\text{ProbNonZeroGradient}(N_i, p_i) = 1 - p_i^{N_i} - (1 - p_i)^{N_i}$ is the probability of obtaining a non-zero gradient (see Section 3.2) for GRPO, and $\text{InfoGain}(p_i)$ quantifies the informativeness of a gradient if one occurs. It can also be extended to other algorithms; see Appendix E. Our design emphasizes coverage of effective gradients across prompts: it accounts for whether a non-zero gradient is likely to appear, but not for the exact balance of positive versus negative samples.

In this work, we define InfoGain as a measure of the expected increase in success probability after a gradient update, while noting that alternative formulations could be explored in future work. Formally, let p_i^t denote the success rate before the update and p_i^{t+1} the rate after the update. We define

$$\text{InfoGain} = \Delta p_i = p_i^{t+1} - p_i^t.$$

Directly computing this requires access to the post-update success probability, which is intractable.

Proposition 1. *With the Taylor expansion, the* InfoGain *can be approximated by* $\boxed{p_i(1-p_i)^2}$.

Please refer to Appendix D for detailed derivation. This modeling is admittedly idealized and not guaranteed to be exact. Nevertheless, it captures key intuitions while remaining simple to implement. Its behavior can be summarized as follows:

- $\text{InfoGain}(p_i)$ is maximized at $p_i = 1/3$. This aligns with the intuition that uncertain-but-promising samples are most valuable.
- $\text{InfoGain}(p_i)$ is asymmetric: for equally distant values of p_i from $1/3$, harder tasks yield larger information gain than easier tasks. Furthermore, $\text{InfoGain}(p_i) \to 0$ as $p_i \to 0$ or $p_i \to 1$, meaning extremely hard or extremely easy prompts provide diminishing value.

We visualize our defined $\text{Value}(N_i, p_i)$ in Figure 4. This contour plot shows lines of equal value, highlighting the interplay between the success rate p_i and the exploration budget N_i. For example, the three highlighted points demonstrate that different combinations of p_i and N_i can yield comparable high values. A task with the success rate $p_i = 0.35$ (which is close to $1/3$), requires a relatively small exploration budget of $N_i = 4$ to achieve a high value. However, for tasks with success rates further from this optimum, such as a harder task with $p_i = 0.19$ or an easier one with $p_i = 0.52$, the required exploration budget is now specified as $N_i = 16$ or $N_i = 8$ respectively to reach the same value level.

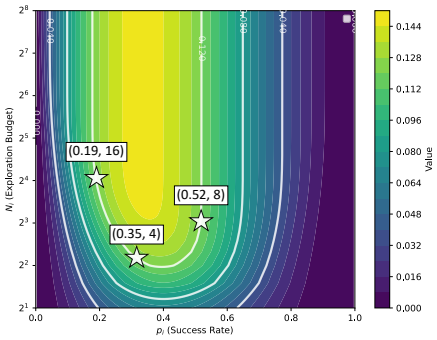


Figure 4: The interplay between success rate, exploration budget and the value.

4.2 Algorithm Implementation

In practice, the success rate p_i is not directly available as a prior and must be estimated from collected samples. In this work, we employ a simple heuristic: using the success rates observed in the previous epoch as estimates for the current one. Specifically, the first epoch may follow a homogeneous budget allocation rule, after which the proposed knapsack-based approach leverages the estimated success rates $\widehat{p}_i$ to guide allocation. Although this strategy introduces some delay and noise, it has proven empirically effective. More sophisticated online estimation techniques (e.g., logistic regression) that account for task correlations present promising directions for future improvement.

These estimated $\widehat{p}_i$ values are directly used to formulate the discrete constrained optimization problem (Equation 5), which can be solved in polynomial time using standard dynamic programming techniques. With Numba (Lam et al., 2015) acceleration, it typically runs within 1–2 seconds.

Overall, our knapsack-based exploration method integrates seamlessly into large-scale RL training pipelines with minimal modifications (see Listing 1 in the Appendix). Computationally, it adds negligible overhead. Algorithmically, it introduces no additional hyperparameters to tune and does not bias policy gradients. From a systems perspective, core components of inference (e.g., vLLM-based accelerated generation (Kwon et al., 2023)) and training (e.g., FSDP (Zhao et al., 2023) and Megatron (Shoeybi et al., 2019)) remain unchanged, ensuring full compatibility with existing infrastructure.

5 EXPERIMENTS

5.1 MAIN RESULTS

Experiment Setting. We implement Knapsack-RL and baseline methods using the large-scale RL training framework `Verl` (Sheng et al., 2025). Our primary focus is GRPO (Shao et al., 2024), a widely examined method, and we refer to our specific implementation as *Knapsack-GRPO*. Training utilizes the DAPO-Math-17K dataset (Yu et al., 2025), which comprises 17,917 prompts, each with a ground truth answer for verification.

We conduct experiments with both pre-trained and instruction-tuned models. The pre-trained models include Qwen3-4B-Base (Yang et al., 2025) and Qwen2.5-Math-7B (Yang et al., 2024). For instruction-tuned models, we utilize DeepSeek-R1-Distill-Qwen-1.5B (Guo et al., 2025) (abbreviated as DPSK-R1-Distill-1.5B) and Qwen3-4B-Instruct-2507 (Yang et al., 2025) (abbreviated as Qwen3-4B-Instruct). In each iteration, we employ a mini-batch size of $M = 256$ prompts and generate $N = 8$ rollouts. Our models are trained for 1,000 iterations. The extensive training duration of 1,000 iterations for Qwen2.5-Math-7B, for example, requires about 1,400 GPU hours with A100 GPUs.

For evaluation, we follow (Luo et al., 2025b) and assess our method on several mathematical reasoning benchmarks: AIME, AMC, MATH, MINERVA, and OLYMPIAD Bench (OLYMPIAD for short). Given AIME's small sample size, we combine its 2024 and 2025 editions into a single dataset, hereafter referred to as AIME. Additionally, we include GPQA (Rein et al., 2023) as an out-of-domain evaluation, which tests scientific reasoning across physics, chemistry, and biology. All reported performance metrics are averaged over 16 generated responses.

Table 1: Evaluation performance (`avg@16`) comparison across different models and benchmarks.

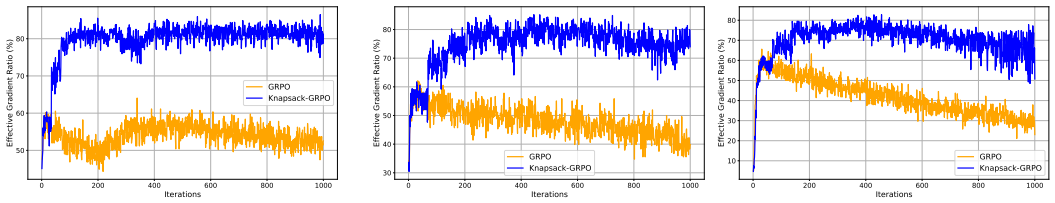
	AIME	AMC	MATH	MINERVA	OLYMPIAD	GPQA	Avg
DPSK-R1-Distill-1.5B	25.3	62.1	81.4	25.8	41.7	39.1	42.9
+ GRPO	27.6	71.1	84.0	27.6	46.4	36.7	45.9
+ Knapsack-GRPO	**34.0**	**75.1**	**86.7**	**28.5**	**49.7**	**40.3**	**49.7**
Qwen3-4B-Base	6.6	29.9	48.0	19.4	23.1	26.4	22.9
+ GRPO	20.7	56.9	80.6	31.9	44.9	**46.6**	43.2
+ Knapsack-GRPO	**20.8**	**66.0**	**81.0**	**35.7**	**46.2**	45.5	**45.1**
Qwen3-4B-Instruct	47.7	82.5	92.4	35.4	61.6	43.0	58.6
+ GRPO	47.0	**84.9**	**92.5**	**41.8**	61.8	54.4	59.2
+ Knapsack-GRPO	**48.2**	83.1	**92.5**	38.2	**63.5**	**59.9**	**61.9**
Qwen2.5-Math-7B	12.3	41.0	61.2	11.8	26.1	22.0	26.7
+ GRPO	23.9	70.6	81.7	33.6	41.9	40.8	45.2
+ Knapsack-GRPO	**24.3**	**77.4**	**83.9**	**34.5**	**44.1**	**43.8**	**47.5**

We report the evaluation performance in Table 1, observing consistent improvements across all tested models after applying our RL training. Specifically, Knapsack-GRPO consistently outperforms GRPO. For instance, in terms of average performance, it improves by 3.8 points for DPSK-R1-Distill-1.5B compared to GRPO. On specific benchmarks, the improvements are even more significant: for example, 6.4 points on AIME for DPSK-R1-Distill-1.5B, 9.1 points on AMC for Qwen3-4B-Base, 5.5 points on GPQA for Qwen3-4B-Instruct, and 6.8 points on AMC for Qwen2.5-Math-7B.

5.2 UNDERSTANDING KNAPSACK-BASED EXPLORATION

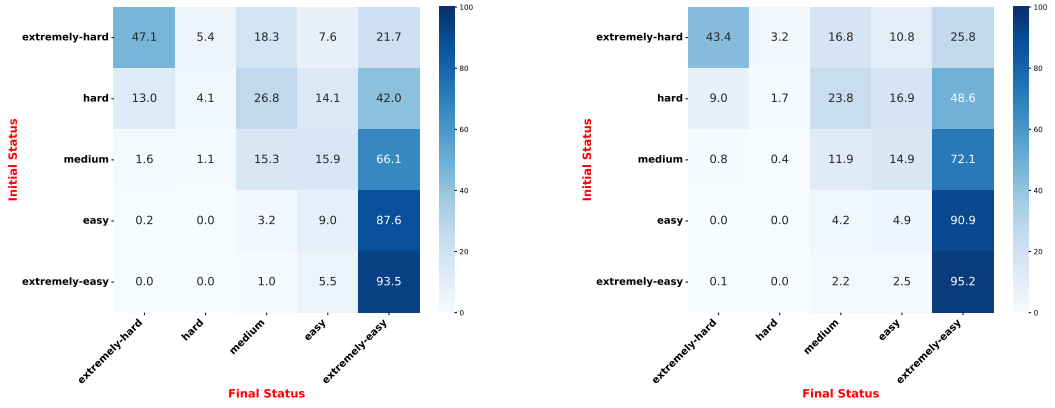
This section delves into understanding the superiority of knapsack-based exploration. We analyze its efficacy through gradient effectiveness and task status dynamics during training, focusing on the Qwen2.5-Math-7B model.

Effective Gradient Ratio. Figure 5 shows the effective gradient ratio during training, as defined in Equation (4). Knapsack-based budget allocation improves this ratio by approximately 20-40% across models. Unlike uniform allocation, the knapsack method avoids a clear decreasing trend. This stems from dynamically distributing exploration budgets, targeting tasks with mixed successful and failed trajectories. These observations partially explain Knapsack-GRPO's policy improvements.



(a) DPSK-R1-Distill-1.5B (b) Qwen3-4B (c) Qwen2.5-Math-7B

Figure 5: Effective gradient ratio during training.



(a) GRPO (b) Knapsack-GRPO

Figure 6: Prompt transition matrices for Qwen2.5-Math-7B during training. The cell (i, j) indicates the percentage of samples transitioning from status i to status j.

Task Transition Dynamics. To understand our method's influence on learning, we analyze prompt evolution during training. We categorize prompts into five performance statuses based on success rate (p_i): `extremely-hard` ($p_i = 0$, all failures), `hard` ($0 < p_i \leq 0.2$), `medium` ($0.2 < p_i < 0.8$), `easy` ($0.8 \leq p_i < 1.0$), and `extremely-easy` ($p_i = 1.0$, all successes). Our analysis covers two aspects: 1) prompt status transitions after training, and 2) final prompt status distribution.

Figure 6 visualizes the 5×5 transition matrix for Qwen2.5-Math-7B training, illustrating prompt category transitions. Knapsack-GRPO demonstrates superior efficiency in learning challenging tasks. Specifically, the self-absorption frequency for `extremely-hard` samples (prompts remaining in that status) is 43.4% for Knapsack-GRPO, notably lower than GRPO's 47.1%. Furthermore, Knapsack-GRPO shows a higher transition rate to `extremely-easy` tasks (last column in heatmap) than GRPO, indicating more effectively mastered samples.

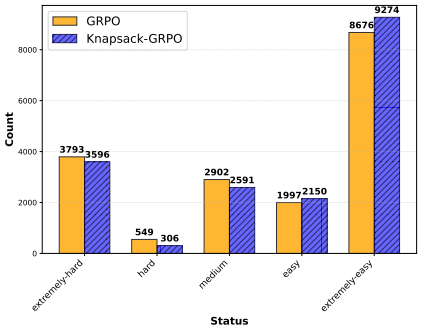


Figure 7: Distribution of sample statuses after training.

We also examine the final distribution of prompt statuses after training, specifically by counting the training samples in each status, as depicted in Figure 7. Knapsack-GRPO has 3,596 `extremely-hard` tasks, less than GRPO's 3,793. This 197-task reduction suggests Knapsack-GRPO's dynamic budget allocation makes them more tractable. Consistent with observed transitions, Knapsack-GRPO yields 9,274 `extremely-easy` tasks, surpassing GRPO's 8,676. Despite these promising results, approximately 20% of prompts remain in the `extremely-hard` category even after 1,000 training iterations. We investigate if these are truly unsolvable: for Knapsack-GRPO, 577 of these challenging prompts recorded at least one positive trajectory during optimization, implying they are not inherently unsolvable. Future research could explore experience replay techniques to address these samples more effectively.

For a more detailed understanding of the Knapsack-GRPO exploration process, additional visualizations are included in Appendix G. These reveal that our knapsack-based method can assign up to 93 exploration budgets to particular tasks, a dynamic allocation that is not computationally intractable with a uniform budget allocation approach.

5.3 Scalability

Finally, we experiment with varying rollout sizes to investigate performance under different computational resource constraints. While our previous experiments focused on a total budget of $N_{\text{total}} = 256 \times 8 = 2048$, here we examine cases with $N = 4$ and $N = 16$, respectively. Note that the parameter N is agnostic to Knapsack-GRPO due to its different allocation strategy.

The results for the Qwen2.5-Math-7B model are shown in Figure 8. KnapSack-GRPO demonstrates clear advantages when computational resources are limited, improving performance from 39.8 to 45.5 in the low-budget setting, while maintaining its superiority even with larger rollouts. Notably, these results indicate that through more efficient exploration budget allocation, KnapSack-GRPO achieves performance levels that would require about 2x the computational resources for standard GRPO to match.

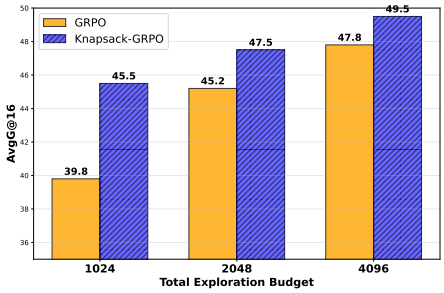


Figure 8: Performance comparison under different exploration budgets.

6 Related Work

Due to space constraints, we discuss works most closely related to ours in the main text and refer to an additional review in Appendix B. Our work is motivated by the intuition that aligning task difficulty with computational resource allocation can improve system efficiency. This concept has been explored in various contexts (Lin et al., 2024; Zhang et al., 2024; Chen et al., 2025b; Zhang et al., 2025b; Wang et al., 2025b; Sun et al., 2025). To name a few, Chen et al. (2025b) proposed leveraging advantage as an estimate of data difficulty to design a curriculum, while Sun et al. (2025) introduced perplexity as a metric for curriculum design. Additionally, Yu et al. (2025) presented the concept of "dynamic sampling" to address the issue of low effective gradients; however, it is crucial to clarify that their "sampling" refers to selecting prompts that yield effective gradients, rather than dynamically allocating exploration budgets.

We remark that these prior works primarily focus on *prompt selection* within frameworks that largely maintain a homogeneous exploration budget. In contrast, our approach operates along a different axis: the dynamic allocation of exploration resources during *response rollouting*. It aims to addresses the need for more extensive exploration on challenging tasks directly. To underscore this fundamental difference, consider that prompt selection methods might prioritize a *subset* of simple prompts to achieve a high effective gradient ratio. Our work, however, aims to dynamically design the exploration budget for *all* prompts to ensure that each receives sufficient exploration to generate effective gradients, especially those that are inherently harder.

7 Conclusion

Motivated by the observation that RL agents require extensive exploration on challenging tasks to gather informative feedback and drive self-improvement, we investigate the problem of optimally allocating computational resources for exploration. We formulate this problem as a knapsack optimization, where each task-budget pair is treated as an item with an associated cost and value. This framework enables us to prioritize harder tasks, thereby yielding more effective gradients and leading to superior policy improvements. This comes at no additional computational cost, effectively offering a "free lunch". We view this work as an initial step toward unlocking RL's potential in LLM post-training through scaling exploration. Looking forward, moving beyond the straightforward stochastic rollout strategy considered here toward richer exploration methods and more structured allocation frameworks presents a promising avenue for future research.

Ethics and Reproducibility Statement

This work primarily focuses on the algorithmic design for allocating exploration budgets within the context of RL training for language models. Our study is purely computational and does not involve human subjects, sensitive data, or any ethically contentious datasets. By enhancing training efficacy, our method aims to reduce overall computational costs and, consequently, mitigate the carbon footprint associated with large-scale model development.

To ensure full reproducibility of our findings, we provide comprehensive details regarding the training frameworks, hyper-parameters, and experimental settings in Appendix C and F. Furthermore, we are committed to publicly releasing our code, relevant datasets, and trained models for research purposes.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Anthropic. System card: Claude opus 4 & claude sonnet 4. *https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf*, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Peter Chen, Xiaopeng Li, Ziniu Li, Xi Chen, and Tianyi Lin. Spectral policy optimization: Coloring your incorrect reasoning in grpo. *arXiv preprint arXiv:2505.11595*, 2025a.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for llm reasoning. *arXiv preprint arXiv:2505.14970*, 2025b.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023.

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.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. Treerl: Llm reinforcement learning with on-policy tree search. *arXiv preprint arXiv:2506.11902*, 2025.

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.

Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 312–320. IEEE, 1982.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, et al. Treepo: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling. *arXiv preprint arXiv:2508.17445*, 2025.

Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: a simple, effective, and efficient reinforcement learning method for aligning large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 29128–29163, 2024.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*, 2025.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025a. Notion Blog.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2, 2025b. Notion Blog.

George B Mathews. On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1):486–490, 1896.

AI Meta. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. *https://ai. meta. com/blog/llama-4-multimodal-intelligence/, checked on*, 4(7):2025, 2025.

OpenAI. Gpt-5 system card. *https://openai.com/index/gpt-5-system-card/*, 2025.

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.

David Pisinger and Paolo Toth. Knapsack problems. In *Handbook of Combinatorial Optimization: Volume1–3*, pp. 299–428. Springer, 1998.

D Rein, B Hou, A Stickland, J Petty, R Pang, J Dirani, J Michael, and S Bowman. Gpqa: A graduatelevel google-proof q&a benchmark, nov. *arXiv preprint arXiv:2311.12022*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Yan Sun, Guo Jia, Stanley Kok, ZihWanao Wang, Zujie Wen, and Zhiqiang Zhang. Stretching the comfort zone: Boost data efficiency for rl training with prepo!, August 2025. URL https://yansun-x.notion.site/data-efficiency-prepo.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Peng-Yuan Wang, Tian-Shuo Liu, Chenyang Wang, Yi-Di Wang, Shu Yan, Cheng-Xing Jia, Xu-Hui Liu, Xin-Wei Chen, Jia-Cheng Xu, Ziniu Li, et al. A survey on large language models for mathematical reasoning. *arXiv preprint arXiv:2506.08446*, 2025a.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Every rollout counts: Optimal resource allocation for efficient test-time scaling. *arXiv preprint arXiv:2506.15707*, 2025b.

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.

Xumeng Wen, Zihan Liu, Shun Zheng, Zhijian Xu, Shengyu Ye, Zhirong Wu, Xiao Liang, Yang Wang, Junjie Li, Ziming Miao, et al. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *arXiv preprint arXiv:2506.14245*, 2025.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jiarui Yao, Yifan Hao, Hanning Zhang, Hanze Dong, Wei Xiong, Nan Jiang, and Tong Zhang. Optimizing chain-of-thought reasoners via gradient variance minimization in rejection sampling and rl. *arXiv preprint arXiv:2505.02391*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Chuheng Zhang, Wei Shen, Li Zhao, Xuyun Zhang, Xiaolong Xu, Wanchun Dou, and Jiang Bian. Policy filtration for rlhf to mitigate noise in reward models. *arXiv preprint arXiv:2409.06957*, 2024.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, et al. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*, 2025a.

Kexun Zhang, Shang Zhou, Danqing Wang, William Yang Wang, and Lei Li. Scaling llm inference efficiently with optimized sample compute allocation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7959–7973, 2025b.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

A Use of LLMs

The drafting of this manuscript was enhanced through the use of a large language model, which assisted in grammatical refinement and the optimization of content organization.

B Additional Related Work

Our work is motivated by a fundamental challenge in LLMs: models are trained with multiple, heterogeneous data sources. Specifically, we study in the axis of data difficulty and note that the difficulty of prompts varies significantly and, more critically, this difficulty evolves and changes across training iterations. This type of data heterogeneity in RL for LLMs has been previously recognized. Notably, Li et al. (2024) observed substantial variations in reward distributions across different prompts, which posed significant challenges for stable gradient estimation (specifically, in terms of variance). Their approach primarily focused on mitigating these issues through new baseline designs, largely operating within the *exploitation* stage of RL by refining how models learn from collected data. Following this, many advanced policy optimization approaches have been proposed (e.g., (Shao et al., 2024; Ahmadian et al., 2024; Yu et al., 2025)). We refer readers to the recent surveys (Zhang et al., 2025a; Wang et al., 2025a) and references therein for an overview. In contrast, our work directly addresses the *exploration* challenge introduced by this data heterogeneity, focusing on how to efficiently gather the most informative data from the outset.

During the preparation of our paper, we noticed a concurrent work by Yao et al. (2025) that also investigates dynamic resource allocation. However, their approach primarily operates within the framework of rejection sampling and RAFT (Dong et al., 2023), focusing on minimizing stochastic gradient variances. In contrast, our study directly addresses online RL settings, employing a knapsack-based design to explicitly balance exploration costs with the potential value derived from each task.

Our method aligns with the principle of test-time scaling (Snell et al., 2024; Brown et al., 2024), which allocates more computational resources (e.g., via best-of-N sampling or majority voting) to identify superior responses. Thus, our approach shares the objective of scaling response generation to enhance the quality of identified solutions. Additionally, our work aligns with the broader strategy of scaling computational resources in post-training to unlock performance in downstream tasks (Jaech et al., 2024; Liu et al., 2025).

Finally, we focus on the straightforward strategy of on-policy exploration with independently sampled rollouts, chosen for its simplicity and tractability. While our method successfully emphasizes allocating additional resources to difficult tasks, some tasks remain unsolved. For these, more advanced exploration strategies may be required beyond simple *independent* rollouts from the initial prompt. In particular, we see strong potential in tree-based exploration methods, inspired by the efficiency of Monte Carlo Tree Search in the challenging task AlphaGo (Silver et al., 2016). Within the context of LLMs, techniques such as *rollbacks* to advantageous intermediate states—as recently explored in Tree-RL (Hou et al., 2025) and TreePO (Li et al., 2025)—are especially promising. We believe that integrating such intelligent state-restarting mechanisms with our framework for optimal computational resource allocation represents a natural and impactful direction for future research.

C Implementation

In this section, we provide more details in implementing Knapsack-based exploration.

Handling Extreme Cases. Our value function defined in Section 4.1 assigns a zero value to prompts with empirical success rates of 0 or 1, which would otherwise lead to zero budget allocation for these prompts. To prevent their complete exclusion and maintain coverage:

- For $\widehat{p}_i = 1.0$ (prompts always solved correctly), the estimate may be not accurate from history samples, so we allocate a small minimum budget (e.g., 2) to ensure they are still considered. This can be achieved by set N_{low} in Equation (5).
- For $\widehat{p}_i = 0.0$ (prompts never solved correctly), we employ a fallback allocation strategy. We first estimate the total budget required for prompts with $p_i \in (0, 1]$ according to Theorem 1 and the above rule. Any remaining budget is subsequently distributed among extremely hard tasks. This strategy is particularly beneficial in later training stages where many prompts become easy, thus freeing up capacity to focus on hard tasks.

Rollout Balancing. In practice, the total number of trajectories ($M \times N$) is typically generated by W parallel workers (where $W < M$), often leveraging efficient inference engines like vLLMs (Kwon et al., 2023). While a homogeneous allocation rule allows for a simple division of M prompts among W workers (each performing N rollouts per prompt), our knapsack-based approach can lead to significant imbalance in allocated rollouts per prompt. This occurs because certain prompts may be allocated disproportionately large exploration budgets, creating an uneven workload and potentially leading to GPU idles and inefficient resource utilization.

To address this issue, we employ a simple rollout balancing strategy: we treat each allocated rollout for a prompt as an individual execution job. These execution jobs are then randomly dispatched to the available workers, with the inference engine generating one response per prompt. This approach is suitable for settings where prompts are not excessively long, thus not strictly requiring advanced techniques like prefix caching. For scenarios involving longer prompts, we would consider using the Karmarkar–Karp bin-packing algorithm (Karmarkar & Karp, 1982) to group prompts into approximately balanced batches based on their allocated budgets. Workers would then process these balanced groups of prompts, potentially utilizing prefix caching.

Listing 1: Python pseudo code implementation of knapsack RL. Two components are modified: (1) budget allocation is replaced with knapsack optimization for better resource distribution, and (2) task status is updated based on external feedback.

```
def budget_allocation(batch, total_budget, **kwargs):
-    budget = np.full(len(batch), total_budget // len(batch['prompt']))
+    budget = knapsack(batch['status'], total_budget, **kwargs)
    indices = []
    for task_id, task_budget in enumerate(budget):
        if task_budget > 0:
            indices.extend([task_id] * task_budget)
    return batch.select_idxs(indices)

gen_batch = budget_allocation(batch, total_budget, **kwargs)
if rollout_balancing:
    indicies = np.random.shuffle(np.arange(len(batch['prompt'])))
    batch = batch.select_idxs(indicies)
batch = actor.generate_sequences(gen_batch)
batch = compute_rewards_and_advantages(batch)
train_dataset.update_status(batch)
actor.update(batch)
```

D PROOF

Proof of Theorem 1. We prove both parts of the lemma.

Part 1: High probability bound. We want to find the minimum N such that $\mathbb{P}(g_i \neq 0) \geq \alpha$ for a given $\alpha \in (0, 1)$. From the problem setup, we have:

$$\mathbb{P}(g_i \neq 0) = 1 - p_i^N - (1 - p_i)^N,$$

For the condition $\mathbb{P}(g_i \neq 0) \geq \alpha$ to hold, we require:

$$1 - p_i^N - (1-p_i)^N \geq \alpha$$
$$p_i^N + (1-p_i)^N \leq 1 - \alpha.$$

Let $q = \max\{p_i, 1-p_i\}$. Since $p_i \in (0,1)$, we have $q \geq \frac{1}{2}$. Without loss of generality, assume $p_i \geq \frac{1}{2}$, so $q = p_i$ and $1 - p_i \leq p_i$. The case $p_i < \frac{1}{2}$ follows by symmetry.

Since $(1-p_i) \leq p_i$, we have $(1-p_i)^N \leq p_i^N$ for $N \geq 1$. Therefore:

$$p_i^N + (1-p_i)^N \leq 2p_i^N = 2q^N.$$

For large N, the term q^N dominates $(1-q)^N$ since $q > \frac{1}{2}$. More precisely, we have:

$$\lim_{N\to\infty} \frac{(1-q)^N}{q^N} = \lim_{N\to\infty} \left(\frac{1-q}{q}\right)^N = 0, \tag{6}$$

since $(1-q)/q < 1$.

Therefore, for sufficiently large N, the constraint (6) is dominated by the term q^N:

$$q^N \lesssim 1 - \alpha \quad \Longleftrightarrow \quad N \ln q \lesssim \ln(1-\alpha). \tag{7}$$

Since $q < 1$, we have $\ln q < 0$, which gives:

$$\boxed{N \gtrsim \frac{\ln(1-\alpha)}{\ln q} = \frac{\ln(1-\alpha)}{\ln(\max\{p_i, 1-p_i\})}}.$$

Part 2: Expected number of rollouts (rigorous proof). Let $X_1, X_2, \ldots$ be i.i.d. Bernoulli random variables with $\Pr(X_i = 1) = p \in (0,1)$, where 1 denotes "success" and 0 denotes "failure". Define

$$N^{\text{first}} \equiv N = \min\{n \geq 1 : \text{both 0 and 1 have appeared among } X_1, \ldots, X_n\}.$$

We compute $\mathbb{E}[N]$ by conditioning on the first trial X_1.

Case 1: $X_1 = 1$ (probability p). After the first success, we still need to wait until the first failure occurs. The waiting time for the first failure follows a geometric distribution with success probability $1-p$, whose expectation is $1/(1-p)$. Thus

$$\mathbb{E}[N \mid X_1 = 1] = 1 + \frac{1}{1-p}.$$

Case 2: $X_1 = 0$ (probability $1-p$). By symmetry, we wait for the first success; its waiting time has expectation $1/p$, so

$$\mathbb{E}[N \mid X_1 = 0] = 1 + \frac{1}{p}.$$

Applying the law of total expectation:

$$\begin{aligned}\mathbb{E}[N] &= p\Big(1 + \frac{1}{1-p}\Big) + (1-p)\Big(1 + \frac{1}{p}\Big) \\ &= 1 + \frac{p}{1-p} + \frac{1-p}{p} \\ &= \frac{1}{p} + \frac{1}{1-p} - 1.\end{aligned}$$

Hence, the expected number of rollouts until we first observe both a success and a failure is

$$\boxed{\mathbb{E}[N^{\text{first}}] = \frac{1}{p} + \frac{1}{1-p} - 1}.$$

This completes the proof of the second part of Lemma 1. □

Proof of Proposition 1. We provide a rigorous derivation under the following assumptions:

- The policy follows a softmax distribution: $p_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$ for action k.
- The gradient update follows the policy gradient rule with advantage A:

$$z_k \leftarrow z_k + \eta \cdot A \cdot \mathbb{I}[k = y] \cdot \nabla_{z_k} \log p_y \tag{8}$$

  where η is the learning rate and y is the chosen action.
- We assume unit learning rate ($\eta = 1$) and unit advantage ($A = 1$) for simplicity.

Step 1: Taylor expansion. For small parameter changes, the change in success probability can be approximated by:

$$\Delta p_y \approx \sum_{k=1}^{K} \frac{\partial p_y}{\partial z_k} . \Delta z_k$$

Step 2: Computing partial derivatives. For the softmax probability $p_y = \frac{\exp(z_y)}{\sum_{j=1}^{K} \exp(z_j)}$, we have:

$$\frac{\partial p_y}{\partial z_y} = p_y(1 - p_y), \quad \text{and} \quad \frac{\partial p_y}{\partial z_k} = -p_y p_k, \quad \text{for } k \neq y.$$

Step 3: Determining parameter updates. Under the policy gradient update rule, we have:

$$\nabla_{z_k} \log p_y = \mathbb{I}[k = y] - p_k.$$

Therefore, the parameter updates are:

$$\Delta z_y = \mathbb{I}[y = y] - p_y = 1 - p_y,$$
$$\Delta z_k = \mathbb{I}[k = y] - p_k = 0 - p_k = -p_k, \quad \text{for } k \neq y$$

Step 4: Computing InfoGain. Substituting the partial derivatives and parameter updates:

$$\begin{aligned}
\Delta p_y &= \frac{\partial p_y}{\partial z_y} \Delta z_y + \sum_{k \neq y} \frac{\partial p_y}{\partial z_k} \Delta z_k \\
&= p_y(1 - p_y) \cdot (1 - p_y) + \sum_{k \neq y} (-p_y p_k) \cdot (-p_k) \\
&= p_y(1 - p_y)^2 + p_y \sum_{k \neq y} p_k^2
\end{aligned}$$

Step 5: Simplification under first-order approximation. For the first-order Taylor approximation to be accurate, we require small parameter updates. Under this condition, the cross-terms $\sum_{k \neq y} p_k^2$ are second-order in the update magnitude and can be neglected compared to the main term $p_y(1 - p_y)^2$.

Therefore, we obtain:

$$\text{InfoGain} \approx \boxed{p_y(1 - p_y)^2}.$$

This completes the proof.

□

To validate this approximation, we conduct an empirical study with 100 actions, comparing the `InfoGain` computed through exact gradient updates against our theoretical approximation from Proposition 1. As shown in Figure 9, the two curves align closely across different success rates, demonstrating that our formula $p(1 - p)^2$ provides a reliable approximation for practical use.

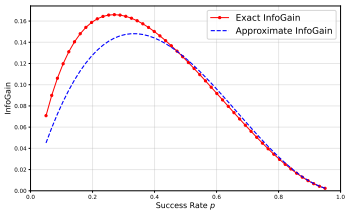


Figure 9: Comparison of exact `InfoGain` and approximate formula.

E Extensions

In this work, we mainly focus on the widely used GRPO (Shao et al., 2024) algorithm to design the optimal allocation strategy. Here we discuss possible extensions for other RL algorithms by adapting the core framework while maintaining the same task

value function structure:

$$\text{Value}(N_i, p_i) = \text{ProbNonZeroGradient}(N_i, p_i) \times \text{InfoGain}(p_i).$$

The key difference lies in how we compute $\text{ProbNonZeroGradient}(N_i, p_i)$ for different algorithms:

- **RLOO** (Ahmadian et al., 2024). RLOO's policy gradient estimator is equivalent to GRPO up to constants, thus we may not need fundamental changes. The probability of obtaining a non-zero gradient remains:

$$\text{ProbNonZeroGradient}(N_i, p_i) = 1 - p_i^{N_i} - (1 - p_i)^{N_i}.$$

- **ReMax** (Li et al., 2024). ReMax leverages the reward of greedy response as baseline, rather than the averaged reward used in GRPO. In this setting, a gradient update occurs only when the sampled trajectory differs from the greedy response. If we denote the probability of the greedy response as α, then the probability of sampling a trajectory different from the greedy response is $1 - \alpha$. The probability of obtaining a non-zero gradient with N_i samples becomes:

$$\text{ProbNonZeroGradient}(N_i, \alpha) = 1 - \alpha^{N_i}.$$

  This represents the probability that at least one of the N_i sampled trajectories differs from the greedy response, thereby producing a gradient signal.

- **REINFORCE** (Williams, 1992). There is no baseline design in vanilla REINFORCE. We can directly calculate the $\text{ProbNonZeroGradient}$ to account for the case where at least one trajectory receives a positive reward:

$$\text{ProbNonZeroGradient}(N_i, p_i) = 1 - (1 - p_i)^{N_i}.$$

  This formulation is simpler than GRPO since we only need to ensure at least one successful trajectory occurs, rather than balancing positive and negative samples.

The proposed framework's modularity allows for straightforward adaptation to other RL algorithms by: (1) identifying the algorithm's gradient computation mechanism, (2) determining conditions for non-zero gradients, (3) calculating the corresponding $\text{ProbNonZeroGradient}$ function, and (4) maintaining the same $\text{InfoGain}(p_i) = p_i(1 - p_i)^2$ formulation across algorithms. This demonstrates the general applicability of our value-based budget allocation approach beyond the specific GRPO implementation.

F Experiment Details

Our experiments utilized the large-scale RL training framework `Verl`, specifically version 0.5.0. No modifications were made to the core training and inference code, with the exception of the advantage calculation, where values were clipped between -5 and 5. This was implemented because, as rollout responses were scaled, we observed their values could become significantly large in extreme cases, thus requiring this additional clipping for numerical stability. Additional implementation details on handling extreme cases and ensuring rollout efficiency are provided in Appendix C.

Following recommendations from (Yu et al., 2025), the learning rate was set to 10^{-6}, with importance sampling clipping ratios (high/low) of 0.28 and 0.2, respectively. Neither KL nor entropy regularization was employed. Models were trained with a maximum sequence length of 4K tokens, with the exception of DPSK-R1-Distill-1.5B, which utilized 8K tokens to accommodate its typically longer Chain-of-Thought (CoT) behaviors requiring more context.

For evaluation results reported during training, models were assessed every 10 training iterations using 16 generated responses. To manage evaluation time, 100 evaluation samples were randomly selected from benchmarks when the total number of samples exceeded this number.

For the final evaluation performance presented in Table 1, different maximum sequence lengths were used to prevent response truncation: 4K tokens for Qwen2.5-Math-7B, 8K tokens for Qwen3-4B and Qwen3-4B-Instruct, and 16K tokens for DPSK-R1-Distill-1.5B. Consequently, these results may not perfectly align with those reported in the training curves.

G Additional Results

G.1 Visualization of Exploration Process

Exploration Budgets. To illustrate the impact of knapsack-based exploration, we visualize the assigned exploration budgets. Specifically, we quantify the frequency with which different exploration budgets are allocated to individual prompts during the training of Qwen2.5-Math-7B. These results are presented in Figure 10. We observe that, even without introducing additional computational resources, our approach can dynamically assign up to 93 exploration budgets to certain tasks. This level of dynamic, high-budget allocation is impractical to achieve under a conventional homogeneous budget allocation framework.

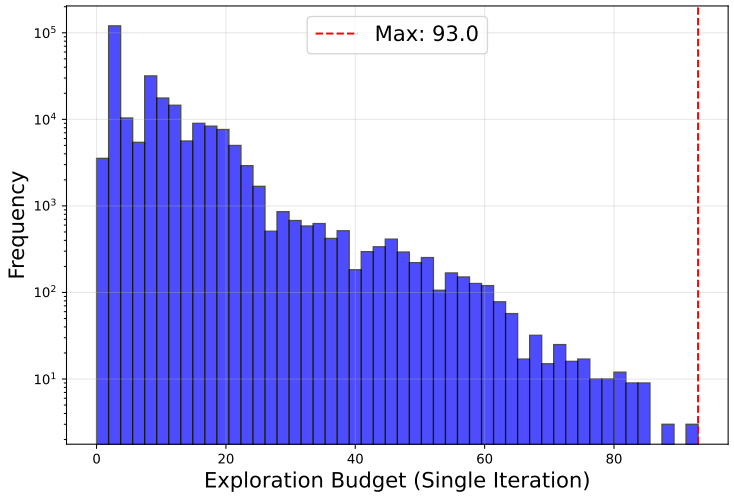


Figure 10: Distribution of exploration budgets allocated by knapsack-based exploration for Qwen2.5-Math-7B during training.

Evolution of Prompts. To illustrate the impact of exploration budgets on individual prompt learning dynamics, we track and visualize the learning trajectories of several randomly selected prompts from the training data in Figure 11. Each subplot corresponds to a unique prompt, identified by its index in the title. We observe that for several examples, our framework effectively allocates more exploration budget, leading to complete learning of the prompt (e.g., prompts in the first row, first column, and second row, first column). Conversely, some tasks remain highly challenging, where neither Knapsack-GRPO nor GRPO achieves satisfactory performance (e.g., the prompt in the third row, second column).

G.2 Training curves.

As references, the training curves for all models are displayed in Figures 12, 13, 14, and 15. Compared with the final results in Table 1, these plots further show that Knapsack-GRPO delivers a rapid performance improvement early in the training process. We also observe a few cases of performance degeneration, which points to the need for exploring more stable policy optimization techniques in future research.

G.3 Ablation Studies

Without Fallback Strategy. In Appendix C, we introduced the *fallback strategy*, which reallocates excess exploration budgets from already-solved prompts to those that remain unsolved. This prevents a common failure mode: difficult prompts may otherwise receive too few resources, while easy prompts are oversampled.

A concrete example is shown in Table 2 with 8 prompts. Without the fallback strategy, the allocation assigns over 50 exploration units to a task with a success rate of 0.9, while the unsolved task (success

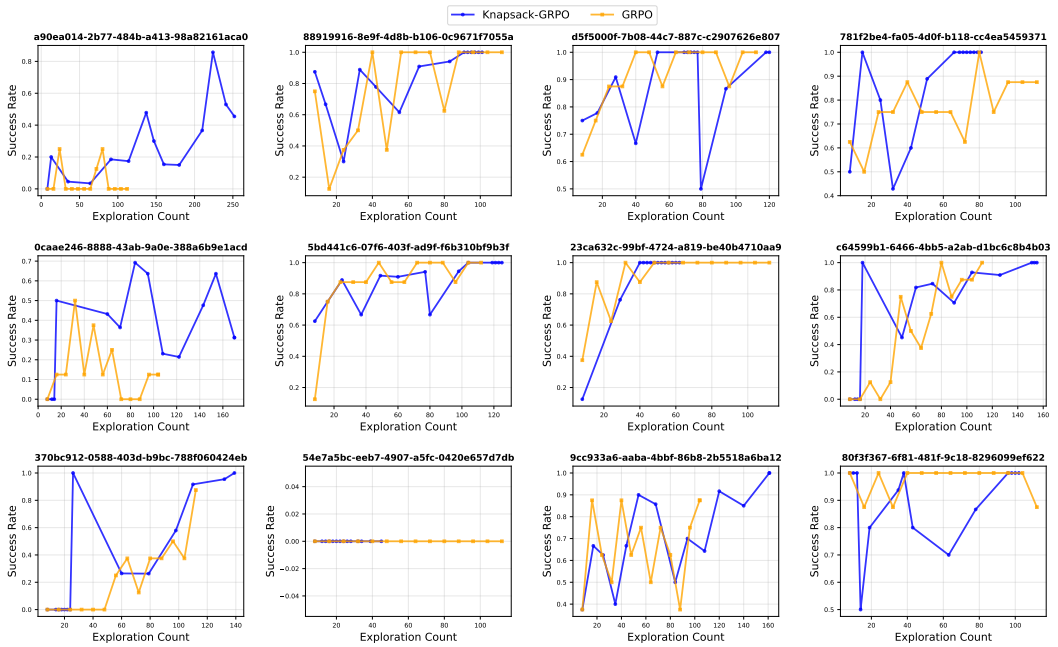


Figure 11: Learning dynamics of randomly selected prompts throughout training, comparing GRPO and Knapsack-GRPO. Each subplot shows the success rate evolution for a specific prompt.

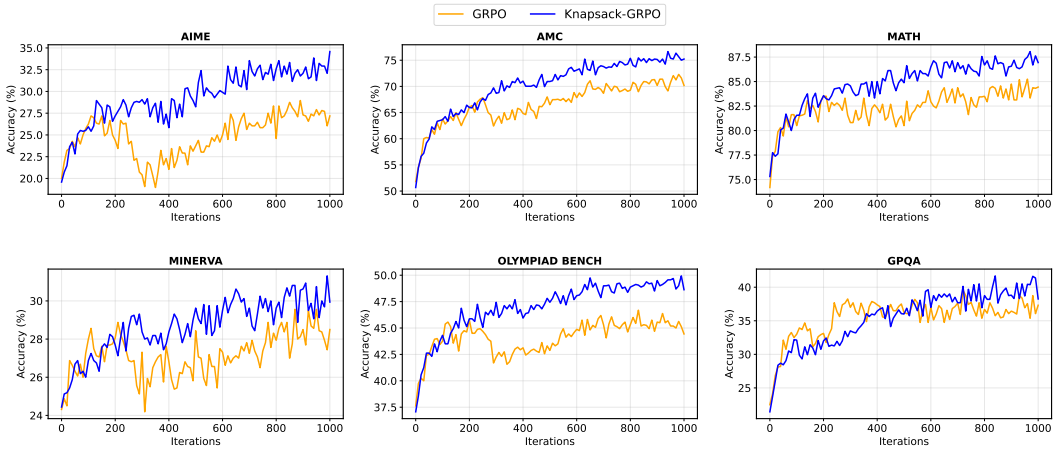


Figure 12: Evaluation performance of DPSK-R1-Distill-1.5B across training iterations.

rate 0.0) receives only 2 units. In contrast, with the fallback strategy, the unsolved task is assigned 29 units—substantially increasing its chance of making progress.

Empirically, this design proves crucial (Figure 16). In our experiments with the Qwen2.5-Math-7B model, removing the fallback strategy led to unstable training, large performance fluctuations on benchmarks such as AMC and OlympiadBench, and overall degraded results. This result suggests that neglecting challenging examples during training weakens the reinforcement signal, ultimately harming the model's ability to generalize.

Low and Up Bounds. Our framework incorporates safeguards in the form of hyper-parameters N_{low} and N_{up}, as defined in Equation (5). N_{up} is set to 128 primarily to facilitate faster computation of the knapsack optimization using dynamic programming; its specific value does not critically impact performance. Conversely, N_{low} is set to 2 to prevent degenerate allocation scenarios, particularly

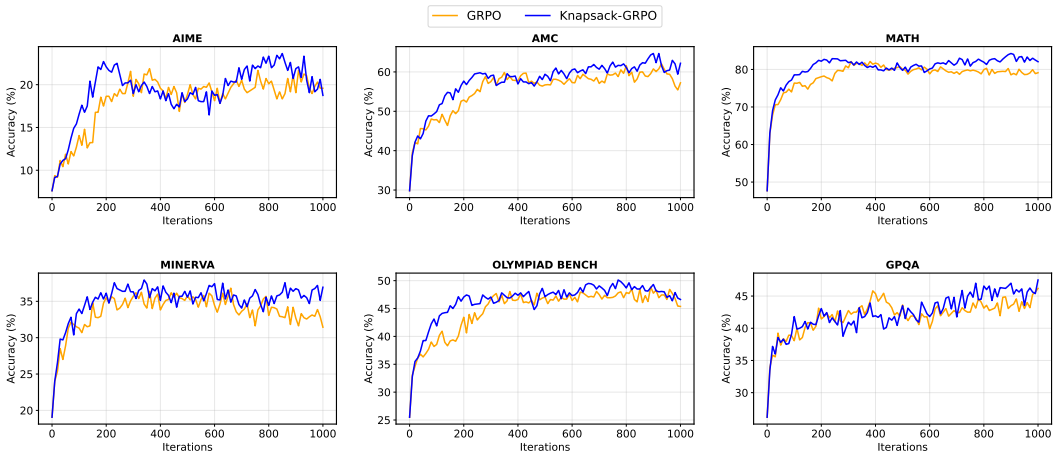


Figure 13: Evaluation performance of Qwen3-4B-Base across training iterations.

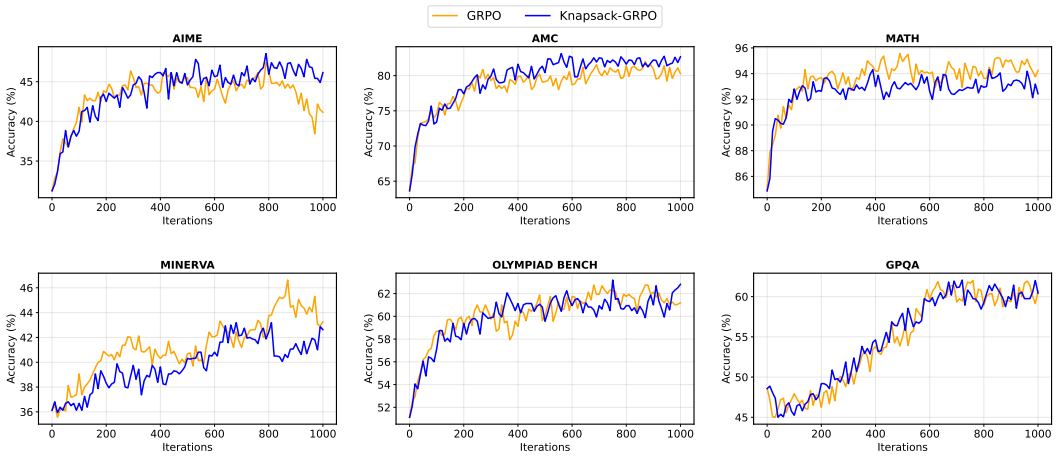


Figure 14: Evaluation performance of Qwen3-4B-Instruct across training iterations.

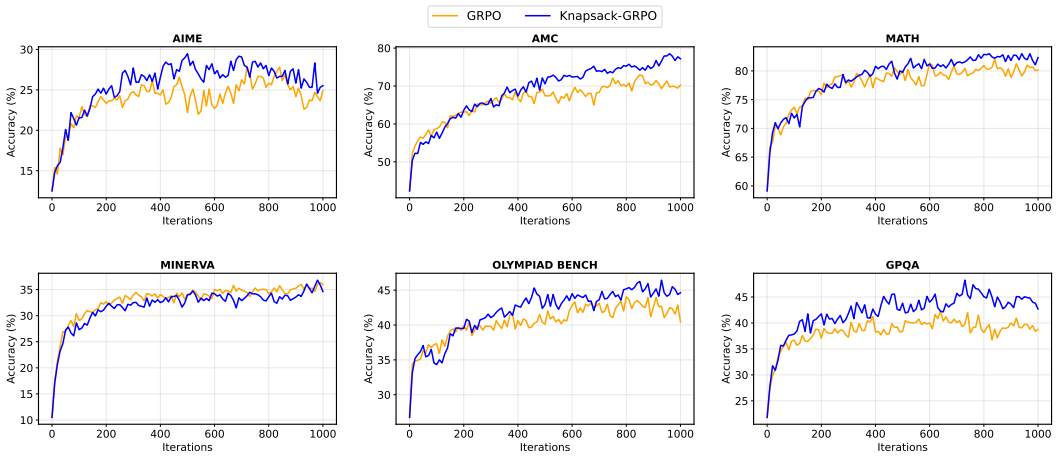


Figure 15: Evaluation performance of Qwen2.5-Math-7B across training iterations.

Table 2: Comparison of budget allocation with and without fallback strategy.

	With Fallback Strategy			Without Fallback Strategy		
Index	**Success Rate**	**Cost**	**Assignment**	**Success Rate**	**Cost**	**Assignment**
1	0.0	∞	29	0.0	∞	2
2	0.9	22	23	0.9	22	50
3	1.0	0	2	1.0	0.0	2
4	1.0	0	2	1.0	0.0	2
5	1.0	0	2	1.0	0.0	2
6	1.0	0	2	1.0	0.0	2
7	1.0	0	2	1.0	0.0	2
8	1.0	0	2	1.0	0.0	2

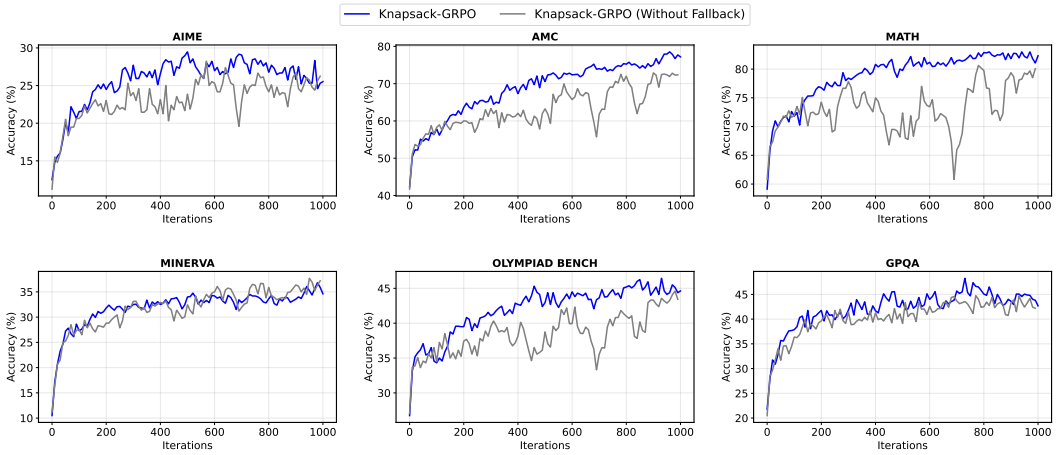


Figure 16: Effect of the fallback strategy. Without it, exploration budgets are disproportionately allocated to prompts with at least one successful trial, while unsolved tasks are largely ignored.

when success rates might be inaccurate, as elaborated in Appendix C. We present ablation results for these bounds in Figure 17, which empirically support these design choices.

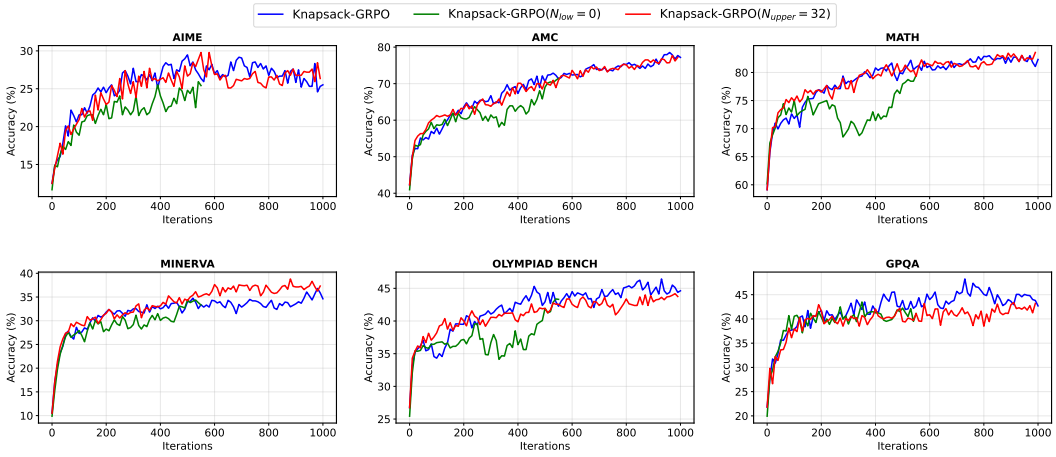


Figure 17: Ablation study on the impact of N_{low} and N_{up} constraints within the knapsack optimization framework.

G.4 Comparing with Dynamic Sampling in DAPO

Dynamic sampling, a technique introduced in the DAPO paper (Yu et al., 2025), selects prompts with a mix of positive and negative rewards, filtering out those with exclusively positive or negative outcomes. This process is repeated until a target number of prompts is accumulated, a strategy that has been shown to be effective.

While effective, dynamic sampling operates on a different principle than our knapsack-based approach. Dynamic sampling aims to scale up effective **prompts**, while our method focuses on scaling up effective **responses**. Since these two approaches are parallel and can be combined, we conducted empirical studies to explore their synergy.

We evaluated the performance of these methods using two different metrics, as shown in the training curves in Figure 18 and Figure 19. Because dynamic sampling requires multiple exploration steps to accumulate enough effective prompts for a single gradient update, we can analyze performance in two ways:

- By exploration budget: Figure 18 shows performance relative to the total number of exploration iterations. This measures how effectively total computation budget is converted into performance gains. We found that dynamic sampling boosts GRPO's performance on benchmarks like AIME and OLYMPIAD, improving the score from 45.2 to 46.2. When we combined dynamic sampling with our knapsack-based exploration, performance on the AMC benchmark improved significantly (from 69.8 to 73.0), resulting in a total performance of 46.5. This is slightly better than dynamic sampling alone but worse than our pure knapsack approach. We attribute this partially to the fact that knapsack-GRPO utilizes more gradient iterations, and therefore do not consider this a negative result.
- By gradient update iterations: Figure 19 displays performance against the number of gradient updates. This metric assesses the value of each gradient update. The results clearly show that effective gradients, whether from dynamic sampling or our knapsack-based exploration, lead to greater performance gains for the same number of update iterations, which validates the core motivation behind both techniques.

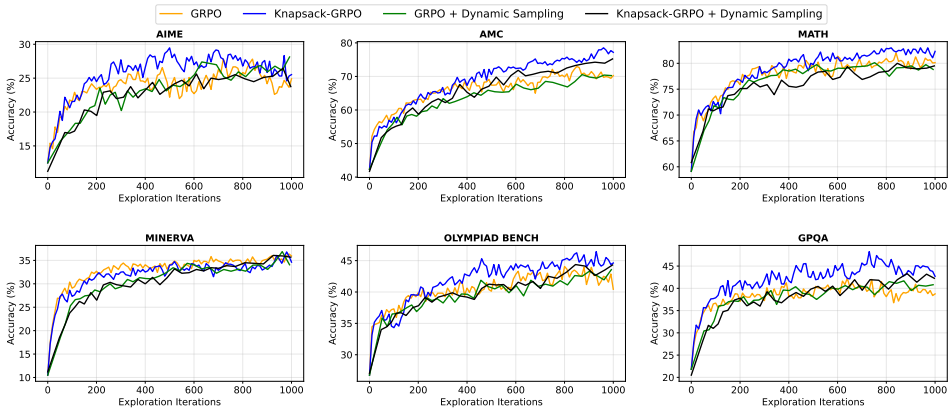


Figure 18: Performance of Qwen2.5-Math-7B relative to the number of exploration iterations, demonstrating how effectively the total computation budget is converted into performance gains.

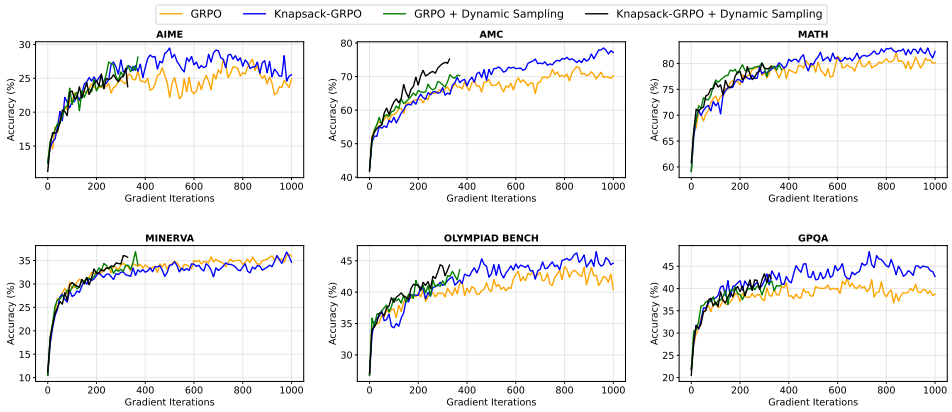


Figure 19: Performance of Qwen2.5-Math-7B as a function of the number of LLM gradient updates. This figure validates that effective gradients, derived from either dynamic sampling or the knapsack-based approach, lead to greater performance gains for the same number of updates.