The Past Is Not Past: Memory-Enhanced Dynamic Reward Shaping

Anonymous ACL submission

Abstract

Despite the success of reinforcement learning for large language models, a common failure mode is reduced sampling diversity, where the policy repeatedly generates similar erroneous behaviors. Classical entropy regularization encourages randomness under the current policy, but does not explicitly discourage recurrent failure patterns across rollouts. We propose **MEDS**, a **M**emory-**E**nhanced **D**ynamic reward **S**haping framework that incorporates historical behavioral signals into reward design. By storing and leveraging intermediate model representations, we capture features of past rollouts and use density-based clustering to identify frequently recurring error patterns. Rollouts assigned to more prevalent error clusters are penalized more heavily, encouraging broader exploration while reducing repeated mistakes. Across five datasets and three base models, MEDS consistently improves average performance over existing baselines, achieving gains of up to 4.13 Pass@1 points and 4.37 Pass@128 points. Additional analyses using both LLM-based annotations and quantitative diversity metrics show that MEDS increases behavioral diversity during sampling.

1 Introduction

With the advancement of the fundamental capabilities of Large Language Models (LLMs), reinforcement learning has achieved significant success across various domains (Shao et al., 2024; Dou et al., 2024; Shojaee et al., 2023). By incorporating reward signals, whether derived from rule-based evaluation (Shao et al., 2024) or proxy models (Ouyang et al., 2022), LLMs iteratively alternate between a sampling phase and a gradient-based optimization phase. As the model's performance is optimized toward maximizing the expected return, designing the reward scoring structure becomes the primary way to guide the model's behavior (Swamy et al., 2025).

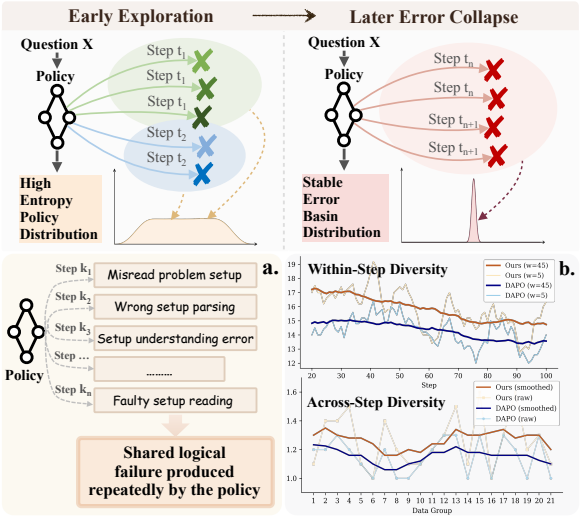


Figure 1: **Top**: During on-policy optimization, the policy evolve from a high-entropy distribution into a narrow, stable error basin. **Bottom left(a)**: Different errors share the same failure mode and become progressively reinforced during training. **Bottom right(b)**: Comparison of output diversity between DAPO and our method, with detailed description provided in Section 4.3.

A well-known failure mode of such on-policy optimization is that, as training progresses, the policy often collapse into a narrow and stereotyped set of behaviors (Yu et al., 2025; Sinha et al., 2026; Chen et al., 2025a). This degeneration produces highly repetitive responses that not only weaken effective exploration and waste on-policy samples, but also entrench the model in self-reinforcing erroneous reasoning trajectories(Zhu et al., 2025a). A series of works (Haarnoja et al., 2018; Wan et al., 2026; Mnih et al., 2016) introduce entropy regularization to promote output diversity by injecting randomness at the level of the current policy distribution. However, in LLM reasoning, such distribution-level stochastic exploration is often insufficient due to the vast action space. Moreover, as illustrated in Figure 1, once training drifts into a particular error pattern, the policy repeatedly re-

produce the same mistake by sampling slightly different reasoning expressions, remaining trapped in this pattern despite continued optimization over many steps.

Inspired by the dynamic reward mechanisms observed in human learning (Holroyd and Coles, 2002), where individuals impose stronger psychological penalties on recurring mistakes to reinforce avoidance, we propose a **M**emory-**E**nhanced **D**ynamic reward **S**haping framework (MEDS). Unlike static reward functions, our method dynamically records historical error patterns and imposes incremental penalties on repetitive failure paths. This mechanism not only facilitates the correction of persistent errors but also encourages the model to escape local optima, thereby enhancing exploration in complex solution spaces. We further provide theoretical analysis showing that additional penalties on repeated errors can improve performance.

In practice, by reusing the layer-wise logits already produced during the forward pass, we construct feature vectors of the model's implicit reasoning trajectory. This reuse introduces almost no additional computational overhead and serves as a lightweight yet effective representation. We then maintain an error memory that is updated online throughout training. By incorporating HDBSCAN-based clustering (McInnes et al., 2017), this memory adaptively captures error types and their density structure, enabling the training objective to adjust penalty strength according to the recurrence of failure patterns.

Empirical results across multiple benchmarks show that our method improves both Pass@1 and Pass@128 across all three models, with gains of up to 4.13 points in Pass@1 and 4.37 points in Pass@128. The largest Pass@128 improvement is observed on Qwen3-8B, where performance increases from 70.81 to 82.67 on a single dataset, corresponding to a relative gain of 17%. We also analyze changes in the model's exploration behavior, demonstrating that our method increases the diversity of exploration. In addition, we calculate the consistency between logit-based clustering and LLM-based clustering, showing that logits can serve as a proxy for reasoning structure. The positive correlation between clustering quality and downstream model performance further validates the effectiveness of our approach. Our main contributions are summarized as follows:

1. To the best of our knowledge, we are the first to introduce memory into reward modeling for LLM reinforcement learning. This approach provides more effective supervision signals during training and mitigates repeated errors. And our method achieves the best performance across multiple datasets and models.

2. We provide a theoretical analysis of the effectiveness of the additional penalty mechanism, offering theoretical support for why our approach improves learning.

3. We introduce a representation-level criterion for logical similarity of response that leverages layer-wise logits as a compact and efficient proxy, incurring little additional computational overhead.

4. We empirically demonstrate that logits are effective clustering features and that our method improves model diversity. Further results show a strong positive correlation between clustering quality and downstream performance, highlighting the importance of recognizing repeated error patterns.

2 Related Work

2.1 Reinforcement Learning with Verifiable Rewards

The application of reinforcement learning in LLMs relies heavily on the design of reward signals. And Reinforcement Learning with Verifiable Rewards (RLVR) has become the dominant paradigm due to the simplicity of its reward signals. Representative works adopt diverse verifiable criteria across tasks. OpenAI (2024) and Shao et al. (2024) assign binary rewards based on exact answer matching. Zhu et al. (2025b) automatically verifies correctness by executing generated programs in a sandbox. Zhang et al. (2025a) uses both temporal localization quality and multiple-choice Video QA accuracy as reward signals. Chen et al. (2025b) employs a multimodal large language model (MLLM) to evaluate similarity between generated and target images. Zhang et al. (2025b) adopts metrics such as nDCG@10 as the primary reward. In all these works, the verifier operates independently within each epoch and assigns rewards based solely on current samples, overlooking useful information in historical trajectories. Our approach instead leverages trajectories from earlier stages to guide later sampling.

2.2 Reward Shaping for Diversity Exploration

During RLVR training, policy updates tend to over-optimize dominant solution patterns, leading to collapse of reasoning diversity (Lightman et al., 2024; Uesato et al., 2022). Traditional approaches mitigate this by constraining policy shift via KL regularization (Ouyang et al., 2022; Schulman et al., 2017) or encouraging randomness via entropy regularization (Haarnoja et al., 2018).

Beyond implicit regularization, several works explicitly reward rare reasoning behaviors. Hu et al. (2026) clusters rollouts by solution strategy and inversely weights advantages by cluster size. Wan et al. (2026) promotes diversity among correct trajectories with token-level entropy regularization. However, these methods operate only on current-batch differences and do not track solution patterns over long-term training history.

To leverage historical information, Song et al. (2025) introduces UCB-style exploration rewards based on historical outcome frequencies. Yet it still lacks fine-grained cross-step reasoning behavior tracking. Another line of work (Li et al., 2024a; Yao et al., 2025; Qu et al., 2025; Hazra et al., 2025; Ma et al., 2024) uses external models to adaptively adjust rewards based on historical performance, but the additional computational cost limits scalability. In contrast, our method captures reasoning patterns via layer-wise logits, enabling history-aware reward adjustment with minimal overhead.

2.3 Inner Thoughts of LLMs

Works that attempt to understand and utilize how models reason and arrive at their final conclusions are an important source of inspiration for our approach. Meng et al. (2022) localizes factual knowledge in the model by examining logit changes. Subsequent studies (Templeton et al., 2024; Olsson et al., 2022; He et al., 2025, 2024b) use Sparse Autoencoders (SAEs) and related logit-intervention techniques to obtain a more fine-grained understanding of LLM reasoning processes. These efforts have further motivated the use of logits for analysis and supervision. For example, Zhao et al. (2025) detects erroneous reasoning by verifying the inner attribute graph, while Tan et al. (2025) leverages layer-wise logits to provide more fine-grained supervision.

A key component of our work is to identify shared patterns across different reasoning trajectories. Inspired by prior work on logit-based model analysis, we cluster logits to capture such patterns without requiring extra models. We describe this approach in detail in Section 3.3.

3 Method

3.1 Preliminaries

Given an input $\mathbf{x} \sim D$, the LLM is modeled as a policy p_θ that generates a response sequence $\mathbf{y} \sim p_\theta(\mathbf{y} \mid \mathbf{x})$, where $\mathbf{y} = \{y_1, y_2, \ldots, y_L\} \in \mathcal{Y}$ represents the complete response and y_t denotes the token generated at step t. The response is evaluated and mapped to a scalar reward via a reward function $r : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. In practical optimization, a KL-regularized objective is adopted, and we maximize the expected return within a trust region. The corresponding training objective is

$$\max_\theta \; \mathbb{E}_{\mathbf{x}\sim D}\Big[\mathbb{E}_{\mathbf{y}\sim p_\theta(\cdot|\mathbf{x})}\big[r(\mathbf{x},\mathbf{y})\big] - \frac{1}{\eta}\,\mathrm{KL}\big(p_\theta(\cdot \mid \mathbf{x}) \,\|\, p_{\mathrm{ref}}(\cdot \mid \mathbf{x})\big)\Big]$$

where p_{ref} is the reference policy that constrains the update space, and η is the regularization coefficient controlling the strength of the constraint.

3.2 Theoretical Benefits of Penalizing Repeated Errors

With the mathematical formulation above, we theoretically analyze the benefits of penalizing repeated errors.

Theorem 1. *Suppose there are two different reward signals, $\mu_1 = r(\mathbf{x},\mathbf{y})$ and $\mu_2 = r(\mathbf{x},\mathbf{y}) - \lambda c(\mathbf{y})$, and let q_1 and q_2 denote the corresponding updated policies under these reward signals. Let $J(q) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim q}[r(\mathbf{x},\mathbf{y})]$. Then it holds that*

$$J(q_2) \geq J(q_1),$$

where λ is the repetition penalty coefficient. $c(\mathbf{y}) : X \times Y \to \mathbb{N}$ denotes the indicator of the number of repeated errors, where a value greater than zero indicates that $\mathbf{y}$ corresponds to an error that has been repeatedly sampled in the past. The function $r(\mathbf{x},\mathbf{y})$ is a task-specific scoring function. The function $c(\mathbf{y})$ increases monotonically with the number of occurrences.

We provide a detailed proof in Appendix B. Intuitively, repeatedly sampled errors indicate higher probability mass under the current policy, and adding an extra penalty helps reallocate this probability. Since the penalty is applied to repeated

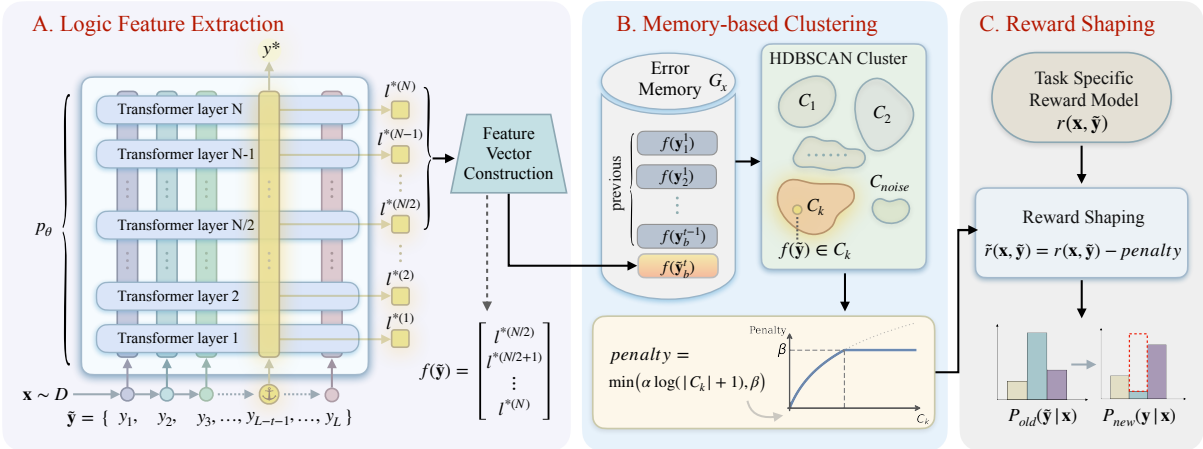


Figure 2: **Overview of MEDS.** The process consists of three stages: **(A) Logic Feature Extraction:** For an input x, we sample a response $\tilde{y}$ and extract a logic feature vector $f(\tilde{y})$ from the layer-wise logits of the first token y^* in the final answer, using the latter half of the Transformer layers to capture reasoning patterns. **(B) Memory-based Clustering:** We maintain a per-prompt error memory G_x and apply HDBSCAN to group historical responses with similar logic features. **(C) Reward Shaping:** For a response assigned to cluster C_k, we compute a penalty $\min(\alpha \log(|C_k|+1), \beta)$ proportional to the cluster size, and adjust the reward as $\tilde{r}(x, \tilde{y}) = r(x, \tilde{y}) - \text{penalty}$, which is then used to update the policy.

errors, the expected return is non-decreasing from the perspective of optimization. **The indicator function acts as a memory for the reward model and adjusts the score based on the historical sampling behavior of the policy.**

3.3 Logic Feature Extraction

When implementing the indicator function $c(\mathbf{y})$, the key aspect is to handle the relationships among different responses. To do so efficiently, we directly reuse the layer-wise logits of each response.

Specifically, suppose the final answer tokens are $\{y_{L-t}, \ldots, y_L\} \subset \{y_1, \ldots, y_L\}$, and let $y^* = y_{L-t}$ denote the first token of the final answer. The corresponding logits at each layer n are $l^{(n)}_{y_{L-t}} \in \mathbb{R}^V$, where V is the size of vocabulary. We select the logit corresponding to this token position, denoted as $l^{*(n)} \in \mathbb{R}$, and aggregate them across layers. The evolution of logits for the final answer depends on feature activations across different layers (Templeton et al., 2024), and therefore contains information about the model's internal reasoning process. Figure 3 provides an intuitive illustration of this process. We also present further examples in Section 4.4, showing that this correlation construction captures more fundamental similarities than approaches based solely on surface-level string matching. The logits vector can be processed in several ways. The two most straightforward approaches are direct aggregation and aggregation after differencing. For clarity, we adopt the direct aggregation approach in our description here and compare the performance of several alternatives in Section 4.6. Since earlier layers typically model simpler semantic information (Cheng et al., 2026), we use the logits from the latter half of the layers. The resulting feature representation is

$$f(\mathbf{y}) = \text{concat}\left(l^{*(n)} \mid n = \frac{N}{2}, \ldots, N\right) \in \mathbb{R}^{\frac{N}{2}}.$$

3.4 Cluster-based Reward Shaping

Based on the constructed response representations, we can directly compute the indicator function and perform the corresponding reward shaping. For each specific prompt $\mathbf{x}$, we maintain a set of feature representations for all responses sampled across historical rollouts:

$$G_{\mathbf{x}} = \{f(\mathbf{y}^1_1), f(\mathbf{y}^1_2), \ldots, f(\mathbf{y}^t_b)\},$$

where b denotes the number of responses sampled for each prompt and t denotes the training epoch. We then apply HDBSCAN clustering on this set to obtain

$$G_{\mathbf{x}} = \{C_1, C_2, \ldots, C_K, C_{\text{noise}}\},$$

where C_i denotes the i-th cluster, and the number of clusters K is determined dynamically.

For a specific response $\tilde{\mathbf{y}}$, let C_k denote the cluster to which $f(\tilde{\mathbf{y}})$ is assigned, where $|C_k|$ denotes the size of the cluster. The corresponding indicator

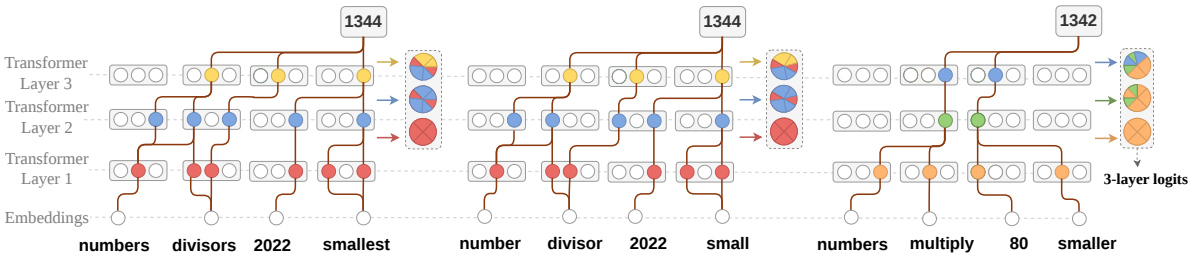


Figure 3: The logits aggregation process, exemplified by a 3-layer model feature graph. Each layer contributes a scalar logit at the final answer token position, whose value is determined by the activated computational circuits. The **left** and **middle** graphs activate nearly identical circuits, resulting in nearly identical 3-layer logit vectors. In contrast, the **right** graph exhibits a substantially different circuit structure, yielding divergent logits.

is defined as $c(\tilde{\mathbf{y}}) = \log(|C_k| + 1)$, a monotonically increasing transformation of cluster size that preserves the ordering used in the theoretical analysis in Appendix B. The reward is then adjusted as follows:

$$\tilde{r}(\mathbf{x}, \tilde{\mathbf{y}}) = r(\mathbf{x}, \tilde{\mathbf{y}}) - \min(\alpha \log(|C_k| + 1), \beta)\,, \quad \text{s.t. } f(\tilde{\mathbf{y}}) \in C_k.$$

where α and β are hyperparameters controlling the penalty strength and the upper bound of the penalty.

4 Experiments

4.1 Experimental Setup

Training Details We train on three base models: Qwen3-1.7B, Qwen3-8B, and Qwen2.5-Math-7B. Qwen3-8B and Qwen2.5-Math-7B represent models with and without explicit reasoning processes, while Qwen3-1.7B serves as a model with a different scale. The training corpus is constructed by merging the DAPO-Math-17K dataset (Yu et al., 2025) with the levels 3-5 of the MATH subdataset (Hendrycks et al., 2021; Liu et al., 2025). All problems are formatted using the Qwen-Math template (Appendix A.1). All reinforcement learning training is implemented based on the *verl* framework (Sheng et al., 2025).

Building on DAPO, our work further introduces a clustering penalty term: for each prompt, we maintain a memory of the logits representations of incorrect samples, where we concatenate the logits from the last 14 Transformer layers and apply L2 normalization, then perform clustering in the logit space using HDBSCAN. During clustering, we set the minimum cluster size to 2, the minimum number of samples to 1, and use Euclidean distance as the distance metric. As for the penalty coefficients, we set $\alpha = 0.1$ and $\beta = 0.2$ for Qwen3-1.7B, Qwen2.5-Math-7B, and set $\alpha = 0.02$ and $\beta = 0.04$ for Qwen3-8B.

In all training runs, we use a prompt-level batch size of 512 and generate 16 rollouts per prompt; the maximum prompt length is 1024 tokens. The maximum response length is 7168 tokens for Qwen3-1.7B, Qwen3-8B, and 3072 tokens for Qwen2.5-Math-7B. All training is conducted on H200 hardware. Other hyperparameters can be found in Appendix A.2.

Baselines We compare the proposed MEDS with two commonly used RL algorithms: GRPO and DAPO. In addition, as discussed in section 2.2, we also compare with a approach that encourages exploration through entropy regularization (Cheng et al., 2025). The hyperparameters and training objectives configurations for these algorithms are provided in appendix A.2. For reference, we also report the performance of the base model without fine-tuning as a baseline.

Evaluation We evaluate our method on five mathematical reasoning benchmarks covering different difficulty levels and problem types: AIME24 (Li et al., 2024b), AMC23 (Li et al., 2024b), MATH500 (Hendrycks et al., 2021), Minerva (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024a).

For pass@k evaluation, we set the number of samples to $n = 128$ and report both pass@1 and pass@128. During evaluation, we also use the Qwen-Math template. The temperature is set to 1.0 and top-p to 1.0 for all models. The random seed is fixed to 0, and the maximum number of tokens is kept consistent with the training configuration.

4.2 Main Results

Table 1 summarizes the performance across three base models and five mathematical reasoning

Method	AIME24		AMC23		MATH500		Minerva		OlympiadBench		Average	
	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128
Qwen3-1.7B	22.58	66.67	58.57	87.50	80.86	95.20	37.82	63.6	41.97	64.15	48.36	75.42
+GRPO	18.20	46.67	51.58	92.50	79.58	95.60	38.62	65.81	40.19	70.37	45.63	74.19
+DAPO	13.46	63.33	57.65	**95.00**	80.25	96.60	39.45	**69.49**	42.20	77.04	46.6	80.29
+GRPO w/ Entropy Adv	12.50	50.00	51.80	**95.00**	78.72	94.00	38.91	62.87	38.47	66.96	44.08	73.77
+MEDS	**23.65**	**70.00**	**65.51**	**95.00**	**84.66**	**97.40**	**41.77**	68.75	**46.86**	**77.63**	**52.49**	**81.76**
Qwen2.5-Math-7B	6.43	56.67	19.82	92.50	30.84	96.00	6.62	63.24	9.27	70.07	14.60	75.70
+GRPO	23.85	66.67	70.08	**97.50**	84.10	**96.20**	39.07	73.16	42.57	73.63	51.93	81.43
+DAPO	32.27	70.00	72.71	**97.50**	85.61	96.00	**43.01**	**75.74**	44.43	75.00	55.61	82.85
+GRPO w/ Entropy Adv	29.74	63.33	70.64	**97.50**	84.50	**96.20**	40.68	70.96	42.57	73.63	53.63	80.32
+MEDS	**34.32**	**76.67**	**74.38**	**97.50**	**86.33**	96.00	42.51	74.26	**44.80**	**75.56**	**56.47**	**84.00**
Qwen3-8B	34.51	70.00	65.72	90.00	83.62	95.40	40.90	61.03	44.69	61.63	53.89	75.61
+GRPO	22.45	70.00	58.79	90.00	82.38	95.40	44.07	64.71	44.22	69.78	50.38	77.98
+DAPO	45.42	73.33	81.37	95.00	89.18	96.60	46.82	65.44	52.77	70.81	63.11	80.24
+GRPO w/ Entropy Adv	27.32	63.33	65.49	92.50	85.89	96.00	45.32	65.44	49.92	70.67	54.79	77.59
+MEDS	**45.78**	**76.67**	**82.62**	**97.50**	**92.51**	**98.20**	**51.58**	**68.02**	**61.12**	**82.67**	**66.72**	**84.61**

Table 1: Main results on five mathematical reasoning benchmarks. Best and second-best are marked with **bold** and underline; MEDS rows are highlighted.

Method	50 steps (min)	100 steps (min)
DAPO	8.00	8.95
MEDS	8.46	9.73

Table 2: Average per-step training time (minutes) of DAPO and MEDS on Qwen2.5-Math-7B, measured over 50 and 100 training steps.

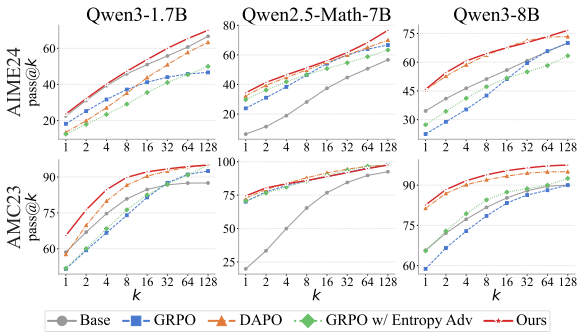


Figure 4: Pass@k results for $k \in [1, 128]$ (log scale). Full results are in Appendix F.

benchmarks. Our method consistently achieves the best overall results.

Across all three model scales, our method consistently achieves the best overall performance, outperforming both the base model and strong baselines such as DAPO. This advantage is observed on both Qwen3-1.7B and Qwen2.5-Math-7B, and becomes even more pronounced on Qwen3-8B model, suggesting that the benefits of our method scales with the capability of the base model. We also plot the pass@k curves, where k takes values in $\{1, 2, 4, 8, 16, 32, 64, 128\}$. The corresponding results are shown in Figure 4, where our method achieves the best performance in most cases.

Table 2 reports the training efficiency comparison. Our method introduces only a small computational overhead compared with DAPO. Considering the consistent performance improvements, this overhead is relatively modest.

4.3 Impact on Exploration Behavior

To better understand how our method influences the model's exploration behavior during reasoning, we conduct a detailed analysis from both behavioral and representational perspectives. Specifically, we first employ an LLM-based evaluation to measure **Within-Step Diversity** and **Across-Step Diversity**, which capture two complementary aspects of rollout diversity. We then further analyze the statistical structure of the logits to provide additional evidence of diversity at the representation level.

Within-Step Diversity measures the diversity of rollouts generated by DAPO and MEDS at the same training step for the same input prompt. Since the first training epoch is mainly used for memory formation and does not involve reward shaping, we restrict this analysis to checkpoints from the second epoch, spanning approximately step 20 to step 100. At each step, we randomly sample 10 groups of 160 rollouts from each method, resulting in 1,600 rollout groups in total, and use *Claude-Haiku-4.5* with the prompt in Appendix D to score group-level output diversity.

Across-Step Diversity instead evaluates whether later-stage rollouts exhibit new reasoning patterns relative to earlier ones for the same model and input prompt. For this purpose, we construct 21 pairs of data groups with varying step intervals, each roughly one epoch apart, where each data group contains 10 rollout groups from DAPO and 10 from MEDS. Using the same prompt-based protocol,

the LLM compares rollouts from the earlier and later steps and scores the extent to which the later outputs introduce novel reasoning patterns. In both settings, higher scores indicate greater diversity.

Beyond these LLM-based evaluations, we further analyze the covariance structure of the stored logits across all steps and compute the ratio between the largest eigenvalue and the sum of all eigenvalues. We refer to this metric as the *Top-1 Eigen Ratio*. Its formal definition is given in Appendix E. A smaller value of this ratio indicates a more uniform spread of the logits across different directions in the representation space.

Our method improves exploration across multiple metrics. The LLM-based evaluation results are shown in the bottom-right panel of Figure 1, and the *Top-1 Eigen Ratio* results are presented in Figure 6 (left). We observe that rollout diversity generally declines as training progresses. However, our method consistently attains higher exploration scores than the baseline under both Within-Step Diversity and Across-Step Diversity. This trend is further supported by the *Top-1 Eigen Ratio*, where our method yields consistently lower values than the baseline, indicating a more diverse exploration behavior.

4.4 Logits can Reflect Reasoning Patterns

To demonstrate that logit vectors capture the underlying logical reasoning structure, we first perform a qualitative analysis of the logit vectors collected from rollouts across the MEDS training process. This preliminary analysis suggests that responses with similar reasoning patterns tend to exhibit similar layer-wise logit trajectories across layers.

Logit patterns reflect reasoning structure Figure 5 presents a case study example. The logits show relatively limited variation in the first 20 layers, where they primarily encode semantic information. In contrast, larger changes emerge in the later layers, where responses with similar reasoning patterns show clear correlations in their logit vectors, with layer-wise values tending to fall within similar ranges. Meanwhile, responses following different reasoning paths remain distinguishable. We further find that logit patterns reflect not only the final answer tokens, but also the intermediate reasoning process. Responses leading to the same answer may exhibit different logit patterns, while responses with different answers may still share similar reasoning behaviors.

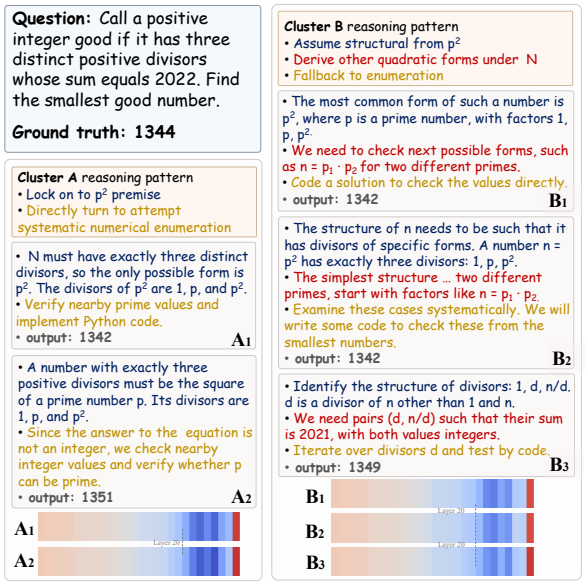


Figure 5: A case study. We manually annotate the reasoning structures of different responses. The responses are grouped into Cluster A containing $\mathbf{A_1}, \mathbf{A_2}$ and Cluster B containing $\mathbf{B_1}, \mathbf{B_2}, \mathbf{B_3}$ by our clustering method. Although $\mathbf{A_1}, \mathbf{B_1}$, and $\mathbf{B_2}$ produce the same incorrect answer 1342, their reasoning structures are different. A similar phenomenon can also be observed for the responses within Cluster B. We further visualize the logits using a heatmap. See Appendix C for the complete case study and the corresponding logit values.

4.5 Logits Provide Correct Clustering Signals

To further validate this observation at scale, we introduce an automated annotation procedure using *Claude-Haiku-4.5* as a proxy annotator. We randomly sample 800 negative responses from the entire dataset. Based on 11 common error categories distilled through manual inspection, we first ask the proxy model to assign each response to one of these error clusters. We then sample response pairs and compare whether the two responses belong to the same cluster according to both the LLM annotations and the logit-based clustering. If the two judgments agree—i.e., both indicate that the responses belong to the same cluster or both indicate that they belong to different clusters—we regard the logit-based clustering as correct for that pair. The agreement rate is then used as the correlation metric. The manually defined error categories and the prompts used for annotation are provided in Appendix D.

Logit clustering serves as an acceptable proxy for LLM-based clustering Figure 6 (right) shows the correlation between our logit-based clustering results and the annotations produced by

Method	AIME24		AMC		MATH500		Minerva		OlympiadBench		Average	
	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128	pass@1	pass@128
DAPO	32.27	70.00	72.71	97.50	85.61	96.00	43.01	75.74	44.43	75.00	55.61	82.85
MEDS-single cluster	29.70	70.00	73.60	100.00	86.60	96.80	41.20	71.69	44.60	74.52	55.14	82.60
MEDS-28-diff	31.88	70.00	72.34	97.50	85.80	96.00	43.38	73.53	45.48	74.37	55.78	82.28
MEDS-14-diff	32.40	73.33	71.64	100.00	86.40	96.60	43.01	72.43	44.15	74.52	55.52	83.38
MEDS-28	35.63	70.00	73.13	97.50	86.20	96.40	41.90	73.16	44.00	73.78	56.17	82.17
MEDS-14	34.32	76.67	74.38	97.50	86.33	96.00	42.51	74.26	44.80	75.56	56.47	84.00

Table 3: Comparison of different clustering feature constructions on Qwen2.5-Math-7B, including assigning all logits to a single cluster (*single cluster*); using adjacent logit differences over 28 layers (*28-diff*) or the last 14 layers (*14-diff*); using logits from all 28 layers; and using only the last 14 layers. The first row corresponds to the vanilla DAPO baseline.

Claude. With a correlation of 61.2%, the logit clustering exhibits a substantial positive relationship with the logical similarity identified by the LLM annotator.

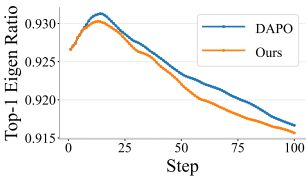
4.6 Clustering Quality Matters

Building on the rich representational signals provided by logits, clustering features can be constructed in multiple ways. We investigate how different feature construction choices affect clustering quality and how these differences further influence downstream evaluation performance.

Specifically, we consider several feature construction strategies that vary in the selected layers and in whether logit differencing is applied. As a worst-case baseline, we also include a degenerate setting in which all responses are assigned to a single cluster, yielding a trivial clustering with virtually no discriminative power. All experiments use Qwen2.5-Math-7B and differ only in the choice of clustering features. Following the Claude-correlation evaluation protocol described in Section 4.4, we report the results of different logit aggregation strategies in Figure 6 (right).

Using the last 14 layers as clustering features yields the best downstream performance. Table 3 reports the results under different feature construction settings. Among all configurations, using the last 14 layers as clustering features yields the best performance. In contrast, excluding logits altogether results in performance worse than DAPO, while every variant that leverages clustering information outperforms the DAPO baseline. This provides further evidence that logits serve as effective historical signals for reward shaping.

Clustering quality strongly correlates with downstream performance. Figure 6 (right) shows that using the last 14 layers as clustering features achieves the highest agreement with the Claude annotations, whereas the degenerate single-cluster setting yields the lowest. More importantly, the ranking of clustering consistency closely mirrors the ranking of downstream performance, suggesting that better clustering yields larger downstream improvements. This provides further evidence for the effectiveness of our approach.



Logit Aggregation	Correlation(%)
single cluster	45.16
28–diff	52.52
14–diff	54.71
28	53.87
14	**61.2**

Figure 6: **(Left)** Top-1 eigen ratio over training steps for DAPO and Ours. Our method consistently yields lower ratios at later steps, indicating less concentration on a single dominant error direction. **(Right)** Correlation of different logit aggregation strategies on Qwen2.5-Math-7B. Single cluster assigns all logits to one cluster; 28 and 14 use direct aggregation over all 28 layers or the last 14 layers; 28–diff and 14–diff use aggregation over layer-wise differences between adjacent logits.

5 Conclusion

In this work, we study the recurring error patterns that emerge during reinforcement learning for LLM reasoning. We introduce MEDS, a memory-aware reward shaping framework that dynamically penalizes recurrent failure patterns. We first provide theoretical analysis demonstrating the effectiveness of our method. By reusing layer-wise logits as lightweight representations of reasoning trajectories, our approach clusters historical responses to identify and suppress repeated error modes. Empirical results across multiple benchmarks show that MEDS consistently improves both reasoning performance and exploration diversity. These findings highlight the importance of incorporating historical trajectory information into reward design and open new directions for improving exploration in LLM reinforcement learning.

Limitations

The main limitation of our study is that the methods explored for utilizing logits are relatively simple and do not incorporate more sophisticated aggregation functions. It remains unclear whether adopting more complex aggregation strategies could further improve performance.

References

Liang Chen, Xueting Han, Qizhou Wang ands Bo Han, Jing Bai, Hinrich Schutze, and Kam-Fai Wong. 2025a. EEPO: exploration-enhanced policy optimization via sample-then-forget. *CoRR*, abs/2510.05837.

Yang Chen, Yufan Shen, Wenxuan Huang, Sheng Zhou, Qunshu Lin, Xinyu Cai, Zhi Yu, Jiajun Bu, Botian Shi, and Yu Qiao. 2025b. Learning only with images: Visual reinforcement learning with reasoning, rendering, and visual feedback. *arXiv preprint arXiv:2507.20766*.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. 2025. Reasoning with exploration: An entropy perspective. *ArXiv*, abs/2506.14758.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. 2026. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *CoRR*, abs/2601.07372.

Shihan Dou, Yan Liu, Haoxiang Jia, Limao Xiong, Enyu Zhou, Wei Shen, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Xuanjing Huang, and Tao Gui. 2024. Stepcoder: Improve code generation with reinforcement learning from compiler feedback. *CoRR*, abs/2402.01391.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. 2025. Revolve: Reward evolution with large language models using human feedback. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024a. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3828–3850. Association for Computational Linguistics.

Zhengfu He, Wentao Shu, Xuyang Ge, Lingjie Chen, Junxuan Wang, Yunhua Zhou, Frances Liu, Qipeng Guo, Xuanjing Huang, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. 2024b. Llama scope: Extracting millions of features from llama-3.1-8b with sparse autoencoders. *CoRR*, abs/2410.20526.

Zhengfu He, Junxuan Wang, Rui Lin, Xuyang Ge, Wentao Shu, Qiong Tang, Junping Zhang, and Xipeng Qiu. 2025. Towards understanding the nature of attention with low-rank sparse decomposition. *CoRR*, abs/2504.20938.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Clay Holroyd and Michael Coles. 2002. The neural basis of human error processing: Reinforcement learning, dopamine, and the error-related negativity. *Psychological Review*, 109:679–709.

Zhiyuan Hu, Yucheng Wang, Yufei He, Jiaying Wu, Yilun Zhao, See-Kiong Ng, Cynthia Breazeal, Anh Tuan Luu, Hae Won Park, and Bryan Hooi. 2026. Rewarding the rare: Uniqueness-aware RL for creative problem solving in llms. *CoRR*, abs/2601.08763.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay V. Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. 2024a. Auto mc-reward: Automated dense reward design with large language models for minecraft. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 16426–16435. IEEE.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, and 1 others. 2024b. Numinamath: The largest public dataset

in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. *CoRR*, abs/2503.20783.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2(11):205.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, and 7 others. 2022. In-context learning and induction heads. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

OpenAI. 2024. Openai o1 system card. *CoRR*, abs/2412.16720.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yun Qu, Yuhang Jiang, Boyuan Wang, Yixiu Mao, Cheems Wang, Chang Liu, and Xiangyang Ji. 2025. Latent reward: Llm-empowered credit assignment in episodic reinforcement learning. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 20095–20103. AAAI Press.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. 2023. Execution-based code generation using deep reinforcement learning. *Trans. Mach. Learn. Res.*, 2023.

Abhijeet Sinha, Sundari Elango, and Dianbo Liu. 2026. Expected return causes outcome-level mode collapse in reinforcement learning and how to fix it with inverse probability scaling. *CoRR*, abs/2601.21669.

Yuda Song, Julia Kempe, and Rémi Munos. 2025. Outcome-based exploration for LLM reasoning. *CoRR*, abs/2509.06941.

Gokul Swamy, Sanjiban Choudhury, Wen Sun, Zhiwei Steven Wu, and J. Andrew Bagnell. 2025. All roads lead to likelihood: The value of reinforcement learning in fine-tuning. *CoRR*, abs/2503.01067.

Yuqiao Tan, Minzheng Wang, Shizhu He, Huanxuan Liao, Chengfeng Zhao, Qiunan Lu, Tian Liang, Jun Zhao, and Kang Liu. 2025. Bottom-up policy optimization: Your language model policy secretly contains internal policies. *CoRR*, abs/2512.19673.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, and 3 others. 2024. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275.

Zhongwei Wan, Yun Shen, Zhihao Dou, Donghao Zhou, Yu Zhang, Xin Wang, Hui Shen, Jing Xiong, Chaofan Tao, Zixuan Zhong, and 1 others. 2026. Dsdr: Dual-scale diversity regularization for exploration in llm reasoning. *arXiv preprint arXiv:2602.19895*.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. 2025. Multi-objective evolution of heuristic using large language model. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 27144–27152. AAAI Press.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. DAPO: an open-source LLM reinforcement learning system at scale. *CoRR*, abs/2503.14476.

Haoji Zhang, Xin Gu, Jiawen Li, Chixiang Ma, Sule Bai, Chubin Zhang, Bowen Zhang, Zhichao Zhou, Dongliang He, and Yansong Tang. 2025a. Thinking with videos: Multimodal tool-augmented reinforcement learning for long video reasoning. *CoRR*, abs/2508.04416.

Le Zhang, Bo Wang, Xipeng Qiu, Siva Reddy, and Aishwarya Agrawal. 2025b. REARANK: reasoning re-ranking agent via reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 2458–2471. Association for Computational Linguistics.

Zheng Zhao, Yeskendir Koishekenov, Xianjun Yang, Naila Murray, and Nicola Cancedda. 2025. Verifying chain-of-thought reasoning via its computational graph. *CoRR*, abs/2510.09312.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. 2025a. The surprising effectiveness of negative reinforcement in LLM reasoning. *CoRR*, abs/2506.01347.

Yaoyu Zhu, Di Huang, Hanqi Lyu, Xiaoyun Zhang, Chongxiao Li, Wenxuan Shi, Yutong Wu, Jianan Mu, Jinghua Wang, Yang Zhao, Pengwei Jin, Shuyao Cheng, Shengwen Liang, Xishan Zhang, Rui Zhang, Zidong Du, Qi Guo, Xing Hu, and Yunji Chen. 2025b. Codev-r1: Reasoning-enhanced verilog generation. *Preprint*, arXiv:2505.24183.

Appendix

A Implementation Details

A.1 Qwen-Math Template

All problems in our training and evaluation datasets are processed using the Qwen-Math template, which provides a standardized format for mathematical problem-solving prompts.

```
<|im_start|>system
Please reason step by step, and put your
final answer within \boxed{}.<|im_end|>
<|im_start|>user
{problem}
<|im_end|>
<|im_start|>assistant
```

A.2 Training Hyperparameters and Objectives

All methods are trained with the Adam optimizer at a learning rate of 1×10^{-6}, weight decay 0.1, gradient clipping threshold 1.0, and 10 warmup steps, with a PPO mini-batch size of 16.

GRPO follows the official DeepSeekMath configuration (Shao et al., 2024), with clipping ratio $\epsilon = 0.2$ and KL penalty coefficient $\beta = 10^{-3}$.

DAPO employs decoupled clipping with asymmetric thresholds $\varepsilon_{\text{low}} = 0.2$ and $\varepsilon_{\text{high}} = 0.28$, and removes the KL penalty term entirely.

GRPO w/ Entropy Adv follows Cheng et al. (2025), augmenting the group-relative advantage with a per-token entropy bonus weighted by α.

The training objectives are defined as follows.

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\Bigg[\frac{1}{K}\sum_{i=1}^{K}\min\Bigg(r_i(\theta)A_i, \text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)A_i\Bigg)\Bigg] - \beta\,\mathbb{D}_{\text{KL}}[\pi_\theta \,\|\, \pi_{\text{ref}}] \tag{1}$$

$$\mathcal{L}_{\text{DAPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\left[\frac{1}{K}\sum_{i=1}^{K}\min\left(r_i(\theta)A_i,\right.\right.$$
$$\left.\left.\text{clip}(r_i(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})A_i\right)\right] \tag{2}$$

$$\mathcal{L}_{\text{GRPO+Ent}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\left[\frac{1}{K}\sum_{i=1}^{K}\min\left(r_i(\theta)\tilde{A}_i,\right.\right.$$
$$\left.\left.\text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)\tilde{A}_i\right)\right] - \beta\,\mathbb{D}_{\text{KL}}[\pi_\theta \,\|\, \pi_{\text{ref}}] \tag{3}$$

where the probability ratio, group-relative advantage, entropy-augmented advantage, and per-token entropy are respectively defined as:

$$r_i(\theta) = \frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)} \tag{4}$$

$$A_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^K)}{\text{std}(\{R_j\}_{j=1}^K)} \tag{5}$$

$$\tilde{A}_i = A_i + \alpha \cdot \mathcal{H}_t \tag{6}$$

$$\mathcal{H}_t = -\sum_{v\in\mathcal{V}} \pi_\theta(v \mid q, o_{<t}) \log \pi_\theta(v \mid q, o_{<t}) \tag{7}$$

B Theoratical Proof

Theorem 2. *Suppose there are two different reward signals, $\mu_1 = r(\mathbf{x},\mathbf{y})$ and $\mu_2 = r(\mathbf{x},\mathbf{y}) - \lambda c(\mathbf{y})$, and let q_1 and q_2 denote the corresponding updated policies under these reward signals. Let $J(q) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim q}[r(\mathbf{x},\mathbf{y})]$. Then it holds that*

$$J(q_2) \geq J(q_1), \tag{8}$$

where λ is the repetition penalty coefficient. $c(\mathbf{y}) : \mathbf{x} \times Y \to \mathbb{N}$ denotes the indicator of the number of repeated errors, where a value greater than zero indicates that $\mathbf{y}$ corresponds to an error that has been repeatedly sampled in the past. The function $r(\mathbf{x},\mathbf{y})$ is a task-specific scoring function.

Proof. For clarity we work pointwise in $\mathbf{x}$ and assume $\mathcal{Y}$ is finite (the continuous case follows by replacing sums with integrals). According to the design that imposes additional penalties on repeated errors, we have:

$$c(\mathbf{y_1}) \geq c(\mathbf{y_2}) \implies r(\mathbf{x},\mathbf{y_1}) \leq r(\mathbf{x},\mathbf{y_2}). \tag{9}$$

That is, responses with larger repetition count are not higher-scoring under the task reward.

For any scoring signal $\mu(\mathbf{x},\mathbf{y})$, define the KL-regularized one-step update (for each $\mathbf{x}$) as

$$q_\mu(\cdot \mid \mathbf{x}) \in \arg\max_{q(\cdot|\mathbf{x})} \Big\{\mathbb{E}_{\mathbf{y}\sim q(\cdot|\mathbf{x})}[\mu(\mathbf{x},\mathbf{y})] - \frac{1}{\eta}\text{KL}(q(\cdot \mid \mathbf{x}) \,\|\, p_{\text{ref}}(\cdot \mid \mathbf{x}))\Big\}.$$

Using a Lagrange multiplier for the constraint $\sum_{\mathbf{y}} q(\mathbf{y} \mid \mathbf{x}) = 1$, the optimizer has the Gibbs form(Rafailov et al., 2023)

$$q_\mu(\mathbf{y} \mid \mathbf{x}) = \frac{p_{\text{ref}}(\mathbf{y} \mid \mathbf{x})\exp(\eta\,\mu(\mathbf{x},\mathbf{y}))}{Z_\mu(\mathbf{x})},$$
$$Z_\mu(\mathbf{x}) = \sum_{\mathbf{y}} p_{\text{ref}}(\mathbf{y} \mid \mathbf{x})\exp(\eta\,\mu(\mathbf{x},\mathbf{y})). \tag{10}$$

Let $\mu_1(\mathbf{x},\mathbf{y}) = r(\mathbf{x},\mathbf{y})$ and $\mu_2(\mathbf{x},\mathbf{y}) = r(\mathbf{x},\mathbf{y}) - \lambda c(\mathbf{y})$ with $\lambda > 0$, and denote $q_1 = q_{\mu_1}$, $q_2 = q_{\mu_2}$.

From (10),

$$q_1(\mathbf{y} \mid \mathbf{x}) = \frac{p_{\text{ref}}(\mathbf{y} \mid \mathbf{x})\exp(\eta r(\mathbf{x},\mathbf{y})}{Z_1(\mathbf{x})},$$
$$q_2(\mathbf{y} \mid \mathbf{x}) = \frac{p_{\text{ref}}(\mathbf{y} \mid \mathbf{x})\exp(\eta r(\mathbf{x},\mathbf{y}) - \eta\lambda c(\mathbf{y}))}{Z_2(\mathbf{x})}.$$

Define $w(\mathbf{y}) := \exp(-\eta\lambda c(\mathbf{y})) \in (0,1]$. Then for each fixed $\mathbf{x}$,

$$q_2(\mathbf{y} \mid \mathbf{x}) = \frac{q_1(\mathbf{y} \mid \mathbf{x})\,w(\mathbf{y})}{\sum_{\mathbf{y}'} q_1(\mathbf{y}' \mid \mathbf{x})\,w(\mathbf{y}')} = \frac{q_1(\mathbf{y} \mid \mathbf{x})\,w(\mathbf{y})}{\mathbb{E}_{\mathbf{y}\sim q_1(\cdot|\mathbf{x})}[w(\mathbf{y})]}. \tag{11}$$

Define the per-$\mathbf{x}$ expected reward under policy q as

$$J_{\mathbf{x}}(q) := \mathbb{E}_{\mathbf{y}\sim q(\cdot|\mathbf{x})}[r(\mathbf{x},\mathbf{y})].$$

Using (11), we obtain

$$J_{\mathbf{x}}(q_2) = \sum_{\mathbf{y}} q_2(\mathbf{y} \mid \mathbf{x}) r(\mathbf{x},\mathbf{y}) = \frac{\sum_{\mathbf{y}} q_1(\mathbf{y} \mid \mathbf{x})\,w(\mathbf{y})\,r(\mathbf{x},\mathbf{y})}{\sum_{\mathbf{y}} q_1(\mathbf{y} \mid \mathbf{x})\,w(\mathbf{y})} = \frac{\mathbb{E}_{\mathbf{y}\sim q_1(\cdot|\mathbf{x})}[w(\mathbf{y})\,r(\mathbf{x},\mathbf{y})]}{\mathbb{E}_{\mathbf{y}\sim q_1(\cdot|\mathbf{x})}[w(\mathbf{y})]}. \tag{12}$$

Hence,

$$J_{\mathbf{x}}(q_2) - J_{\mathbf{x}}(q_1) = \frac{\mathbb{E}_{q_1}[w\,r] - \mathbb{E}_{q_1}[w]\mathbb{E}_{q_1}[r]}{\mathbb{E}_{q_1}[w]} = \frac{\text{Cov}_{q_1}(r(\mathbf{x},\mathbf{y}), w(\mathbf{y}))}{\mathbb{E}_{q_1}[w(\mathbf{y})]}. \tag{13}$$

Since $\mathbb{E}_{q_1}[w(\mathbf{y})] > 0$, it suffices to show that $\mathrm{Cov}_{q_1}(r(\mathbf{x}, \mathbf{y}), w(\mathbf{y})) \geq 0$.

Let $Y \sim q_1(\cdot \mid \mathbf{x})$ and $C = c(Y)$. Note that $w(Y) = \exp(-\eta\lambda C)$ is non-increasing in C. By equation (9), $r(\mathbf{x}, Y)$ is also non-increasing in C (equivalently, the conditional mean $\bar{r}_{\mathbf{x}}(k) := \mathbb{E}[r(\mathbf{x}, Y) \mid C = k]$ is non-increasing in k).

Therefore, by Chebyshev's rearrangement inequality,

$$\mathbb{E}[\bar{r}_{\mathbf{x}}(C)\exp(-\eta\lambda C)] \geq \mathbb{E}[\bar{r}_{\mathbf{x}}(C)] \cdot \mathbb{E}[\exp(-\eta\lambda C)]. \tag{14}$$

Using the tower property,

$$\begin{aligned} &\mathbb{E}_{q_1}[r(\mathbf{x}, Y)w(Y)] \\ &= \mathbb{E}[\mathbb{E}[r(\mathbf{x}, Y) \mid C]\exp(-\eta\lambda C)] \\ &= \mathbb{E}[\bar{r}_{\mathbf{x}}(C)\exp(-\eta\lambda C)]. \end{aligned} \tag{15}$$

Similarly,

$$\begin{aligned} \mathbb{E}_{q_1}[r(\mathbf{x}, Y)] &= \mathbb{E}[\bar{r}_{\mathbf{x}}(C)], \\ \mathbb{E}_{q_1}[w(Y)] &= \mathbb{E}[\exp(-\eta\lambda C)]. \end{aligned} \tag{16}$$

Thus, $\mathrm{Cov}_{q_1}(r(\mathbf{x}, Y), w(Y)) \geq 0$, and (13) implies $J_{\mathbf{x}}(q_2) \geq J_{\mathbf{x}}(q_1)$ for all $\mathbf{x}$.

Taking expectation over $\mathbf{x} \sim D$, we obtain

$$\begin{aligned} J(q_2) &= \mathbb{E}_{\mathbf{x}\sim D}[J_{\mathbf{x}}(q_2)] \\ &\geq \mathbb{E}_{\mathbf{x}\sim D}[J_{\mathbf{x}}(q_1)] \\ &= J(q_1). \end{aligned} \tag{17}$$

This completes the proof. □

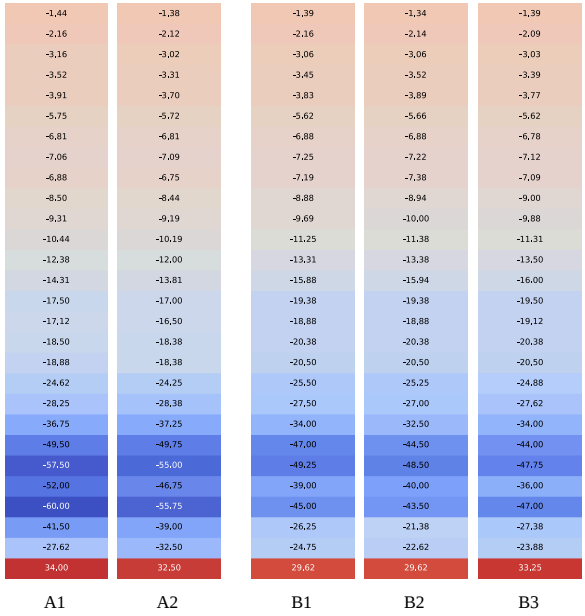


Figure 7: Heatmap in figure 5 with logit values.

C Full Results of Case Study

Question

Call a positive integer *good* if it has three distinct positive divisors whose sum equals 2022. Find the smallest good number.

Ground truth: 1344

C.1 Logit Values

Figure 7 is the logit heatmap in figure 5, which has logit values on it.

C.2 Cluster A Responses

Response A1 *Output: 1342*

To solve the problem of finding the smallest good number, we need to understand that a number is good if it has three distinct positive divisors that sum to 2022.

A key insight here is that if a number n is a good number, then it must have exactly three distinct divisors. The only type of number that has exactly three distinct divisors is a square of a prime number. This is because the number of divisors of a number $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ is given by $(e_1 + 1)(e_2 + 1) \cdots (e_k + 1)$. To have exactly 3 divisors, the only possible form is p^2 because 3 can only be factored as $3 = 3 \times 1$.

The divisors of p^2 are $1, p, p^2$. We need these to sum to 2022, so we need:

$$1 + p + p^2 = 2022$$

This simplifies the equation to:

$$p^2 + p + 1 = 2022$$

Rearranging the terms, we get a standard quadratic equation:

$$p^2 + p - 2021 = 0$$

We will solve this quadratic equation using the quadratic formula $p = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$:

$$\begin{aligned} p &= \frac{-1 \pm \sqrt{1 + 4 \times 2021}}{2} \\ &= \frac{-1 \pm \sqrt{8085}}{2} \end{aligned}$$

We are only interested in the positive value of p, so we need to check if $\sqrt{8085}$ is an integer. We will calculate $\sqrt{8085}$ and check its value. If 8085 is a perfect square, we can proceed further.
By calculating, we find that $\sqrt{8085} \approx 89.9166$, which is not an integer. So we need to find the closest prime number. We check the closest perfect square value and determine:

$$89^2 = 7921 \quad \text{and} \quad 90^2 = 8100$$

Since 8085 lies between these two, we check the values for the primes near 89 and verify. We check whether we can substitute 89 into our equation to see if we get a prime number for p:
If we let $p = 89$: $89^2 + 89 + 1 = 7921 + 89 + 1 = 8011 \neq 2022$.
So we will try $p = 83$: $83^2 + 83 + 1 = 6889 + 83 + 1 = 6973 \neq 2022$.
This also does not match. We will check $p = 83$. Let's use a closest prime number which might satisfy, let's take $p = 43$: $43^2 + 43 + 1 = 1849 + 43 + 1 = 1893 \neq 2022$.
This does not match either. Let's check with smaller primes. We will check $p = 41$: $41^2 + 41 + 1 = 1681 + 41 + 1 = 1723 \neq 2022$.
This is still not 2022. Let's check with $p = 47$: $47^2 + 47 + 1 = 2209 + 47 + 1 = 2257 \neq 2022$.
This also does not work. Now we check $p = 43$: $43^2 = 1849$, $1849 + 43 + 1 = 1893 \neq 2022$.
Let's check with larger prime numbers and verify by code to ensure we get the exact result. We can implement a Python code to check the values and find the smallest n.
We will start by checking $p = 45$ as a possible value: $45^2 = 2025$, which is greater than 2022, but let's proceed and confirm by code.
We need to find p such that $p + 1 + p^2 = 2022$. Let's implement this code to find the exact prime number.

```
import math

def find_smallest_good_number():
    # We will start with the smallest possible
    # number and check upwards
    num = 1
    while True:
        divisors = [d for d in range(1, \
                    num + 1) if num % d == 0]
        if len(divisors) >= 3:
            divisors = sorted(divisors)[:3]
            if sum(divisors) == 2022:
                return num
        num += 1

smallest_good_number = \
                find_smallest_good_number()
print(smallest_good_number)

Output: 1342
```

The smallest good number, which satisfies the condition that three distinct positive divisors sum to 2022, is $\boxed{1342}$.

Response A2 *Output: 1351*

To find the smallest number with exactly three distinct positive divisors that sum up to 2022, we need to analyze the structure of such a number. A number with exactly three positive divisors must be the square of a prime number p, because its divisors are 1, p, and p^2.
Given the condition, if $n = p^2$, then we need $1 + p + p^2 = 2022$. We can set up the equation:

$$p^2 + p + 1 = 2022.$$

Rearranging gives:

$$p^2 + p - 2021 = 0.$$

This is a quadratic equation of the form $p^2 + p - 2021 = 0$. We can solve for p using the quadratic formula $p = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$, where $a = 1$, $b = 1$, and $c = -2021$. This gives:

$$\begin{aligned} p &= \frac{-1 \pm \sqrt{1^2 - 4 \cdot 1 \cdot (-2021)}}{2 \cdot 1} \\ &= \frac{-1 \pm \sqrt{1 + 8084}}{2} \\ &= \frac{-1 \pm \sqrt{8085}}{2}. \end{aligned}$$

We are only interested in the positive root, so we compute:

$$p = \frac{-1 + \sqrt{8085}}{2}.$$

We need to check if $\sqrt{8085}$ is an integer, and if $\frac{-1+\sqrt{8085}}{2}$ is a prime number.
First, let's approximate $\sqrt{8085}$: $\sqrt{8085} \approx 89.9166$. So, $\sqrt{8085}$ is very close to 90. Let's check if 90 is indeed the square root: $90^2 = 8100$, which is slightly larger than 8085. So, $\sqrt{8085} \approx 89.9166 \approx 90$ (not exact, so we check 89).
Let's check 89 directly: $89^2 = 7921$, which is smaller than 8085. Let's check 90: $90^2 = 8100$, which is slightly larger than 8085. Thus, 89 is closer and we need to check it more precisely:

$$89^2 = 7921 \quad \text{and} \quad 90^2 = 8100.$$

Since 89^2 is closer to 8085, we use it: $\sqrt{8085} \approx 89.9166$, and thus:

$$p = \frac{-1 + 89.9166}{2} \approx \frac{88.9166}{2} \approx 44.4583,$$

which isn't an integer. Let's check the closest integer values. We need p to be an integer and prime. Let's check if 89 is a solution by substituting it back:

$$p^2 + p - 2021 = 89^2 + 89 - 2021 = 7921 + 89 - 2021 = 4078 \neq 0.$$

Thus, we check the next closest prime number, and we know that p must be an integer. So let's check manually for the factorization and see if we can find a prime p:
If we try solving the equation directly for integer values:

$$p^2 + p - 2021 = 0 \implies p(p + 1) = 2021.$$

We look for pairs of factors of 2021. We factorize 2021:

$$2021 = 43 \times 47.$$

We see that 43 and 44 don't work, but checking 43 directly in the equation: $43^2 = 1849$ and $43 + 1849 = 1892 \neq 2022$.
Let's check if there's a smaller p that works by iterating through possible prime values. We use Python to automate checking for the smallest n. We'll start by finding potential p values that make $1 + p + p^2 = 2022$ a valid equation and check if p is prime.

```
import math

def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def find_smallest_good_number():
    target_sum = 2022
    p = 1
    while True:
        p_squared = p * p
        if 1 + p + p_squared == target_sum:
            if is_prime(p):
                return p_squared
        p += 1

smallest_good_number = \
                find_smallest_good_number()
print(smallest_good_number)

Output: 1351
```

The smallest number that has exactly three distinct positive divisors summing up to 2022 is $\boxed{1351}$.

C.3 Cluster B Responses

Response B1 *Output: 1342*

To find the smallest positive integer n that has exactly three distinct positive factors whose sum equals 2022, we need to analyze the structure of the factors.
Analysis:
1. Understanding the Factors: For a number n to have exactly three distinct positive factors, the most common form of such a number is p^2, where p is a prime number. The factors of p^2 are $1, p$, and p^2.
So, if $n = p^2$, the sum of the factors $1 + p + p^2 = 2022$. Therefore, we have:

$$p^2 + p + 1 = 2022 \implies p^2 + p - 2021 = 0$$

2. Solving the Quadratic Equation: We solve for p using the quadratic formula $p = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ where $a = 1, b = 1$, and $c = -2021$:

$$\begin{aligned} p &= \frac{-1 \pm \sqrt{1 + 4 \cdot 2021}}{2} \\ &= \frac{-1 \pm \sqrt{8085}}{2} \end{aligned}$$

Since p is a prime number, we need to check if $\sqrt{8085}$ is an integer and if $\frac{-1+\sqrt{8085}}{2}$ is a prime number. We find that $\sqrt{8085} = 89.917$. This means the closest integer value for the square root is 89. Checking if 89 works:

$$p = \frac{-1 + 89}{2} = \frac{88}{2} = 44$$

But 44 is not a prime number. We need to check the next possible forms. Another form for having three factors might be when $n = p_1 \cdot p_2$ for two different primes p_1 and p_2. However, since $p_1 \cdot p_2$ would suggest factors $1, p_1, p_2$, and $p_1 p_2$, having exactly three factors requires a form without additional factors. Instead, let's examine the simplest form more closely. Since 44 wasn't a prime, we incrementally check the next prime values for p:
Iterating through Prime Numbers:
We should check if the factor forms might lead to other structures that fit, or directly check if smaller values fit our sum of three factors form directly.
Let's code a solution to check the values directly: iterate through numbers calculating their factors and check the sum.
Python Code: We will write a Python script to find the smallest number whose factors sum to 2022. We'll start with a function to calculate the sum of the factors and then iterate over potential candidates. Let's implement the solution in Python.

```
import sympy

def sum_of_factors(n):
    # Function to calculate sum of all factors
    # of n
    factors = set(sympy.divisors(n))
    return sum(factors)

# We need to find the smallest n such that
# sum_of_factors(n) == 2022
target_sum = 2022
n = 1

while True:
    if sum_of_factors(n) == target_sum:
        smallest_good_number = n
        break
    n += 1

print(smallest_good_number)

Output: 1342
```

The smallest positive integer n that has exactly three distinct positive factors whose sum equals 2022 is $\boxed{1342}$.

Response B2 *Output: 1342*

To solve the problem of finding the smallest positive integer that has three different positive divisors whose sum equals 2022, we need to delve into the properties of divisors. Suppose $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$ is the prime factor-

ization of n. The total number of divisors of n is given by $(e_1+1)(e_2+1)\cdots(e_k+1)$. If n has exactly three positive divisors greater than 1 and these three divisors sum up to 2022, the structure of n needs to be such that it has divisors of specific forms.

Analysis

1. The number n cannot be a perfect power of more than one prime because it would have more than three divisors. Hence, we must look for specific structures.

2. A number n that is a perfect square of a prime, i.e., $n = p^2$, will have exactly three divisors: 1, p, and p^2. But since we require three different divisors greater than 1, we realize p itself should be considered as having divisors p and p^2 along with a new third different divisor (other than p and p^2). So let's consider the structure that has more than just those simple prime power forms.

Since the simplest structure is a product of two different primes, let's start with factors like $n = p_1 \cdot p_2$. Such n would have divisors 1, p_1, p_2, and p_1p_2. But since we want exactly three different positive divisors (excluding 1) that sum to 2022, let's look into it systematically.

We will check numbers of the form $p^2 \cdot q$, since it would have divisors $1, p, q, p^2, pq, p^2q$. Here we are interested in checking p^2, pq, p^2q or similar other combinations. However, another approach is checking sums systematically for candidate numbers directly.

We will write some code to check these systematically to find the smallest n such that three of its distinct divisors (excluding 1) sum to 2022. The plan is to check each number's divisors and see if any three of them (excluding 1) sum up to 2022. We'll start from the smallest numbers. We will write a function to get the divisors of a number and then check the sums.

```
def get_divisors(n):
    """
    Returns a sorted list of all divisors of n.
    """
    divisors = set()
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            divisors.add(i)
            divisors.add(n // i)
    return sorted(divisors)

def has_three_divisors_summing_to_target(n, \
                                          target):
    """
    Checks if there are three distinct divisors
    (excluding 1) that sum to target.
    """
    divisors = get_divisors(n)
    if 1 in divisors:
        divisors.remove(1)
    for i in range(len(divisors)):
        for j in range(i + 1, len(divisors)):
            for k in range(j + 1, \
                           len(divisors)):
                if divisors[i] + divisors[j] \
                    + divisors[k] == target:
                    return True
    return False

def find_smallest_good_number(target):
    n = 2
    while True:
        if has_three_divisors_summing_to_targ
        et(n, target):
            return n
        n += 1

# We are looking for the smallest number such
# that 3 different divisors (excluding 1) sum
# to 2022
smallest_good_number = \
                find_smallest_good_number(2022)
print(smallest_good_number)
```

```
Output: 1342
```

The smallest positive integer n such that three of its distinct positive divisors (excluding 1) sum to 2022 is $\boxed{1342}$.

Response B3 *Output: 1349*

To solve this problem, we need to find the smallest positive integer n that has three distinct divisors d_1, d_2, and d_3 such that their sum is 2022. Let's break down the problem step by step.

1. Identify the Structure of Divisors:

- For a number n to have three distinct divisors, they must be in the form $1, d, \frac{n}{d}$ where d is a divisor of n other than 1 and n. The sum of these three divisors is $1 + d + \frac{n}{d} = 2022$.
- This equation can be rearranged to: $1+d+\frac{n}{d} = 2022$ or $d + \frac{n}{d} = 2021$. Thus, $n = d(2021 - d)$.

2. Find the Smallest n:

- We need to find pairs $(d, \frac{n}{d})$ such that their sum is 2021 and $d \neq 1, \frac{n}{d}$. Both d and $\frac{n}{d}$ must be integers, so d must be a divisor of n and $2021 - d$ must be an integer greater than d (because $d \neq \frac{n}{d}$).
- The smallest possible d greater than 1 is 2, but let's check all possible values systematically.

We'll iterate over possible divisors d that are less than $\sqrt{2021} \approx 44.95$. Any divisor d must satisfy $d < 1011$ since if $d \geq 1011$, $2021 - d \leq 1010$, and their product would not be n. Let's write a Python code to iterate and find the smallest n.

We'll check each integer d from 2 up to 1010 (since $d \neq 1$ and d cannot be greater than or equal to 1011) to see if $2021 - d$ is an integer and if it's greater than d. For each valid d, we'll compute $n = d \times (2021 - d)$ and check if the sum of the divisors $1 + d + \frac{n}{d} = 2022$. We will then keep track of the smallest n found.

Let's write the Python code to implement this approach:

```
def smallest_good_number(target_sum):
    # Using a large initial value
    smallest_n = float('inf')

    # We start from 2 since 1 is the smallest
    # possible divisor
    for d in range(2, target_sum // 2):
        if (target_sum - d) > d and d * \
                        (target_sum - d) > 0:
            n = d * (target_sum - d)
            if len(divisors(n)) == 3 and \
                sum(divisors(n)) == target_sum:
                smallest_n = min(smallest_n, n)

    return smallest_n

def divisors(n):
    divs = []
```

```
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            divs.append(i)
            if i != n // i:
                divs.append(n // i)
    return sorted(divs)

target_sum = 2022
smallest_n = smallest_good_number(target_sum)
print(smallest_n)

Output: 1349
```

The smallest positive integer that has three distinct positive divisors whose sum is 2022 is $\boxed{1349}$.

D Prompts

Prompt used in Figure 1, Within-Step Diversity

System Prompt:
You are an expert evaluator for AI model outputs. Your task is to assess, for each group of answers independently, how diverse or varied the answers WITHIN that group are. Focus on whether answers in the same group share the same reasoning approach, arrive at the same conclusion, and have similar quality.

User Prompt:

```
## Question
{q_trunc}

---

## Group A — {len(orig_answers)} answer(s)
{fmt_answers(orig_answers)}

---

## Group B — {len(filt_answers)} answer(s)
{fmt_answers(filt_answers)}

---

## Task
For **each group independently**, rate the
INTERNAL DIVERSITY of the answers within that
group on a scale from 1 to 5:

1 = All answers are nearly identical — same
reasoning path, same final answer, just minor
wording differences
2 = Mostly similar — same general approach but
slight variations in steps or details
3 = Moderately diverse — some answers take a
noticeably different approach or differ in
quality
4 = Highly diverse — answers use clearly
different methods, or some are correct while
others are wrong
5 = Extremely diverse — answers are completely
different in approach, quality, or conclusion

Consider for each group:
- Do all answers reach the same final answer?
- Do they all use the same mathematical method
or strategy?
- Is there variation in correctness or
reasoning quality?

Respond with ONLY the following two lines (no
explanation):
Group A Score: <integer from 1 to 5>
Group B Score: <integer from 1 to 5>
```

Prompt used in Figure 1, Across-Step Diversity

System Prompt:
You are comparing two groups of answers generated by the same model at two different training stages for the same math question.

User Prompt:

```
## Question
{q}
---
## Group A (earlier training stage) -
{len(answers_step1)} answer(s)
{fmt(answers_step1)}
---
## Group B (later training stage) -
{len(answers_step2)} answer(s)
{fmt(answers_step2)}
---
## Task
Compare Group B against Group A and judge
whether Group B contains any NEW and DIFFERENT
reasoning paths.
Definition:
- A "reasoning path" means the core line of
reasoning, solution strategy, or analytical
framework used to answer the question.
- Count a path as "new" only if Group B uses a
substantively different reasoning idea that
does not appear in Group A.
- Differences in wording, order, formatting, or
level of detail do NOT count as a new reasoning
path if the core reasoning is the same.
Scoring rule:
- Score 1: Group B has 0 new reasoning paths
compared with Group A
- Score 2: Group B has 1 new reasoning path
compared with Group A
- Score 3: Group B has 2 or more new reasoning
paths compared with Group A
Important:
- Focus only on whether Group B introduces new
reasoning paths relative to Group A.
- Do not score based on correctness, quality,
or final answer unless they reflect a
genuinely new reasoning path.
- Each group has at most 3 answers, so cap the
score at 3.
Respond with ONLY the following line (no
explanation):
Score: <integer from 1 to 3>
```

Prompt used in 4.5

User Prompt:

```
The following answer is wrong. Analyze the
reasoning errors and categorize it into the
following categories:

<error_category_list>
  Completely irrelevant response
  Misinterpretation of problem setup
  Incorrect mathematical modeling
  Unverified assumptions
  Probability and statistics error
  Incorrect use of mathematical theorems /
  formulas / principles
  Errors in basic calculations
  Miscounting error
  Missing cases or edge cases
  Implementation or algorithmic error
  Self-contradiction or abandonment of correct
  intermediate result
</error_category_list>

<question> {question} </question>
<ground_truth> {ground_truth} </ground_truth>
<answer> {answer} </answer>

Output format:
<result>
  <mistake_analysis>
    Step-by-step analysis of the reasoning
    process and where it goes wrong.
  </mistake_analysis>
  <error_category>
    One of the categories in the error category
    list.
  </error_category>
</result>
</think>
```

E Top-1 Eigen Ratio

We provide a formal definition of the *Top-1 Eigen Ratio*. Let $\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$ denote the covariance matrix of the stored logits, and let its eigenvalues be ordered as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$. The *Top-1 Eigen Ratio* is defined as

$$\text{Top-1 Eigen Ratio} = \frac{\lambda_1}{\sum_{j=1}^{d} \lambda_j}.$$

Intuitively, λ_1 measures how much variance in logit space is explained by the dominant principal direction, while $\sum_{j=1}^{d} \lambda_j$ corresponds to the total variance.

F Full results of Pass@k

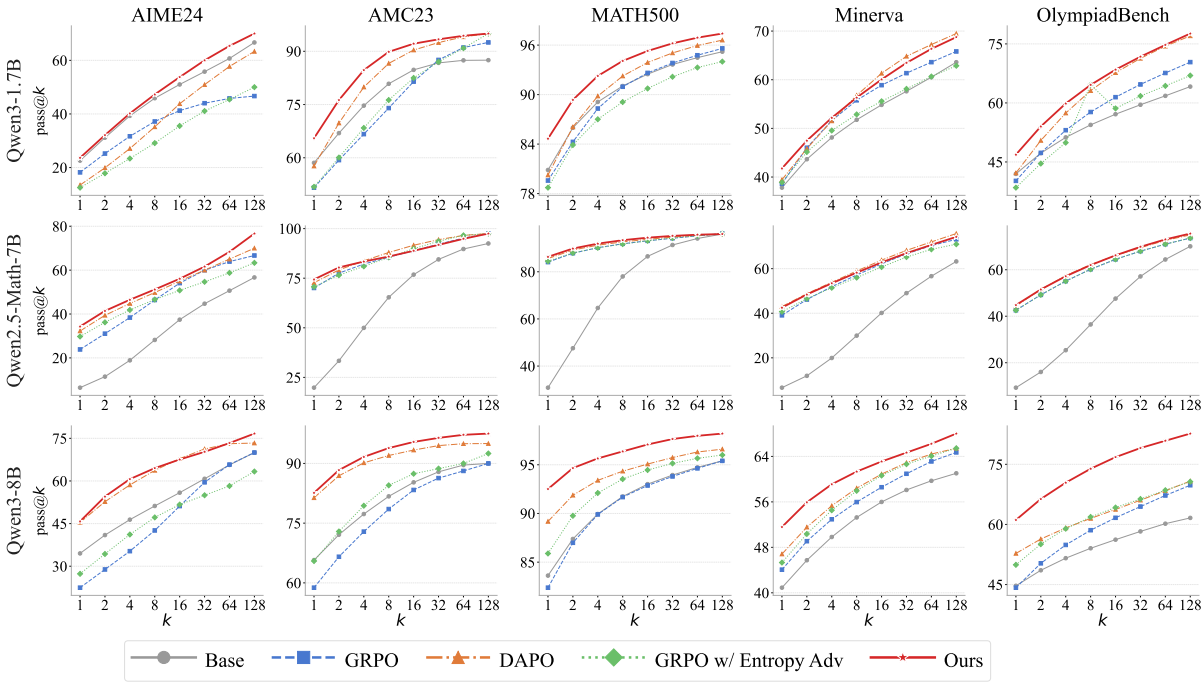


Figure 8: Pass@k across $k \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ on five math benchmarks for three model scales. Each row corresponds to one model; each column to one dataset. Ours consistently matches or outperforms Base, GRPO, DAPO, and GRPO with entropy advantage across settings.