ByteDance | Seed

PAG: Multi-Turn Reinforced LLM Self-Correction with Policy as Generative Verifier

Yuhua Jiang[1,*], **Yuwen Xiong**[2], **Yufeng Yuan**[2], **Chao Xin**[2], **Wenyuan Xu**[2], **Yu Yue**[2], **Qianchuan Zhao**[1], **Lin Yan**[2]

[1]Tsinghua University, [2]ByteDance Seed

[*]Work done at ByteDance Seed

Abstract

Large Language Models (LLMs) have demonstrated impressive capabilities in complex reasoning tasks, yet they still struggle to reliably verify the correctness of their own outputs. Existing solutions to this verification challenge often depend on separate verifier models or require multi-stage self-correction training pipelines, which limit scalability. In this paper, we propose Policy as Generative Verifier (PAG), a simple and effective framework that empowers LLMs to self-correct by alternating between policy and verifier roles within a unified multi-turn reinforcement learning (RL) paradigm. Distinct from prior approaches that always generate a second attempt regardless of model confidence, PAG introduces a selective revision mechanism: the model revises its answer only when its own generative verification step detects an error. This verify-then-revise workflow not only alleviates model collapse but also jointly enhances both reasoning and verification abilities. Extensive experiments across diverse reasoning benchmarks highlight PAG's dual advancements: as a policy, it enhances direct generation and self-correction accuracy; as a verifier, its self-verification outperforms self-consistency.

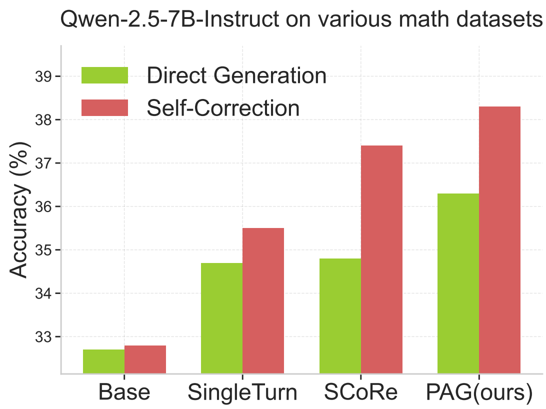
Date: June 3, 2025

Correspondence: jiangyh22@mails.tsinghua.edu.cn, yuwen.xiong@bytedance.com

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable proficiency in solving complex reasoning problems, particularly in domains such as mathematical problem-solving [6, 57] and code generation [16, 17]. Despite these advances, a significant challenge remains: although LLMs are capable of generating complex solutions, they often struggle to reliably verify the correctness of their own reasoning [11, 43]. This verification gap presents a critical obstacle in domains where solution accuracy is essential, such as advanced mathematical reasoning [9, 15, 26], scientific research [21], and dependable decision support systems [38].

To bridge this verification gap, previous research has primarily explored two main directions. The first centers on the use of *standalone verifier models*, which are explicitly trained to evaluate the quality of generated solutions [6, 18, 58, 59]. These verifiers typically rely on human-labeled data or leverage more advanced "LLM-as-a-judge" systems for supervision [46]. While this strategy can be effective, it requires deploying an additional model, thereby increasing both computational costs and system complexity. The second line of work seeks to enhance the self-correction abilities of LLMs themselves. Early studies adopted supervised fine-tuning



SCoRe [13]

Problem. If $n \equiv 2 \pmod 7$, then find the remainder when $(n+2)(n+4)(n+6)$ is divided by 7.

First attempt (incorrect). ... Final Answer: The final answer is 1. I hope it is correct.

Instruction. There might be an error in the solution above because of lack of understanding of the question. Please correct the error, if any, and rewrite the solution.

Second attempt (correct). ... Final Answer: The final answer is 3. I hope it is correct.

Policy as Generative Verifier (PAG)

Problem. Alice and Bob are playing a game. Alice starts first. On Alice's turn, she flips a coin. If she gets a heads, she wins. If not, it becomes Bob's turn. On Bob's turn, he flips a coin. If he gets a tails, he wins. If not, it becomes Alice's turn. What is the probability that Alice wins the game?

First attempt (incorrect). Let's denote the probability that Alice wins the game as P ... the probability that Alice wins the game: $P = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot \left(\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot P\right) \ldots$

Therefore, the probability that Alice wins the game is $\boxed{\frac{1}{3}}$.

Instruction. Check the math solution step-by-step. If you find a mistake: state the wrong step, explain why it's wrong, and end your response with 'The answer is wrong'. If all steps are correct, end your response with 'The answer is correct'.

Self-Verify. Let's break down the probability of Alice winning step by step. ... The answer is wrong.

Instruction. You indicated that your previous answer was wrong. Please provide the correct solution to the math problem.

Second attempt (correct). ... $P = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} P \ldots$

Therefore, the probability that Alice wins the game is $\boxed{\frac{2}{3}}$.

Figure 1 **Top left: PAG** achieves state-of-the-art self-correction performance across diverse mathematical reasoning datasets. **Bottom left:** SCoRe always generates a second attempt regardless of confidence. **Right:** Our PAG framework employs selective revision through self-verification, revising only when the initial attempt is explicitly identified as wrong.

(SFT) on datasets containing errors and their corresponding corrections [4, 23, 57]. However, SFT-based methods often suffer from distribution shift: models trained on static datasets may fail to generalize to new types of errors that arise as the model evolves. As a result, reinforcement learning (RL) has gained traction as a more flexible and adaptive framework [13, 22, 54], allowing models to learn directly from their own, dynamically generated responses.

Nevertheless, effectively leveraging reinforcement learning (RL) for self-correction introduces several challenges. One notable issue is the phenomenon of "model collapse" [13], where the LLM, when prompted to improve its output, may only make minor or superficial changes, or sometimes simply repeat its initial response. This can occur if the RL objective does not sufficiently encourage substantive improvements, or if the model has difficulty distinguishing between genuine corrections and paraphrasing, which can hinder learning progress. To address this, Kumar et al. [13] has explored multi-stage RL training approaches. These approaches often rely on carefully designed reward functions or learning curricula, which can increase the complexity of the training process.

We argue that such complex multi-stage approaches [13] may not be necessary. This paper introduces **Policy as Generative Verifier (PAG)**, a simple yet effective method grounded in a key insight: rather than always generating a second attempt, it is more effective to employ a selective revision mechanism—the model only revises its answer when its own self-verification explicitly identifies an error.

In PAG, the model first acts on the role of a *generative verifier*, critically evaluating its own initial answer. Only when this self-verification step identifies an error does the model proceed to generate a revised response. As illustrated in Figure 1, this "verify-then-revise" workflow fundamentally distinguishes PAG from previous self-correction methods that always perform a second attempt regardless of the model's confidence. This design offers two main advantages. First, by explicitly introducing a verification step, PAG effectively mitigates model collapse, encouraging genuine self-reflection rather than superficial rephrasing of the initial answer. Second, the integrated alternation between generation and verification roles enables the model to jointly improve both its reasoning and verification abilities within a unified training framework.

PAG delivers significant advantages in both training efficiency and self-correction capability. Our framework can be seamlessly integrated with existing instruction-tuned LLMs via multi-turn reinforcement learning. The proposed role-switching mechanism—alternating between policy and verifer—robustly mitigates model collapse, thereby obviating the need for complex multi-stage training protocols or warm-up phases required by prior approaches [13, 48]. Furthermore, PAG's generative verification ability empowers the model to autonomously select the best candidate from multiple self-generated attempts, surpassing majority voting strategies [44], all within a single unified model.

Our main contributions are summarized as follows:

- We propose **Policy as Generative Verifier (PAG)**, a novel and streamlined framework that enables large language models (LLMs) to achieve self-correction by alternately switching between policy and generative verifier roles.
- We demonstrate that PAG can be efficiently implemented using multi-turn reinforcement learning, eliminating the need for complex multi-stage training approaches and significantly simplifying the training pipeline.
- We conduct extensive experiments among Qwen and Llama models show that PAG substantially enhances LLM reasoning capabilities, improving both direct generation accuracy and self-correction performance.
- We further show that PAG demonstrates strong generative verification capabilities, and our self-verification best-of-N strategy is especially effective, achieving better results than majority voting.

2 Related Work

LLM Self-Correction Recent advances [4, 12, 19] have demonstrated that large language models (LLMs) exhibit self-improvement capabilities when provided with external feedback in agentic tasks or code repair tasks. In this work, we focus on intrinsic self-correction, where the model improves its responses during inference without access to any external feedback. The underlying motivation is that LLMs often possess latent knowledge that is difficult to elicit through standard next-token generation, yet self-correction can distill this hidden knowledge into improved outputs [10, 40, 54]. Prior studies [23, 37] have shown that prompting can sometimes elicit self-correction abilities in LLMs; however, such intrinsic self-correction is neither robust nor significant in complex domains [11, 43]. Several approaches [28, 30, 52] have attempted to leverage supervised fine-tuning by distilling reranked or filtered generations from human or stronger model generated data, but the resulting self-correction abilities suffer from distribution shift and limited generalization.

Reinforcement learning (RL) methods have been proposed to address these challenges. SCoRe [13] employs multi-turn RL for self-correction, but discovers that naively applying multi-turn RL leads to model collapse, where the model's second attempt exhibits minimal deviation from the first. To mitigate this issue, they introduce a warm-up phase that leverages RL to train the model to decouple its behavior across the two attempts by optimizing second-attempt accuracy while explicitly constraining the distribution of first attempts to match the base model, followed by multi-turn RL as a second-stage training procedure. Recent concurrent works [22, 48] propose self-rewarding RL frameworks similar to our approach, but they require extensive supervised fine-tuning and only explore single-turn RL. In contrast, our approach jointly trains the policy and verifier via multi-turn RL, and, crucially, does not require any warm-up phase—training can begin directly from existing instruction-tuned models.

Generative Verifier Traditional verifiers are predominantly discriminative, typically trained as classifiers to distinguish between prefer and disprefer responses [6, 18, 41, 59]. In contrast, generative verifiers [24, 58] leverage the text generation capabilities of large language models (LLMs), adopting a next-token prediction paradigm for verification. This approach enables the verifier to perform chain-of-thought reasoning, thereby facilitating more nuanced and effective verification processes. Recent works [5, 20, 36, 50] have begun to explore reinforcement learning (RL) for training generative verifiers. However, most prior efforts focus exclusively on either policy optimization or verifier training in isolation. In this work, we propose a unified RL framework

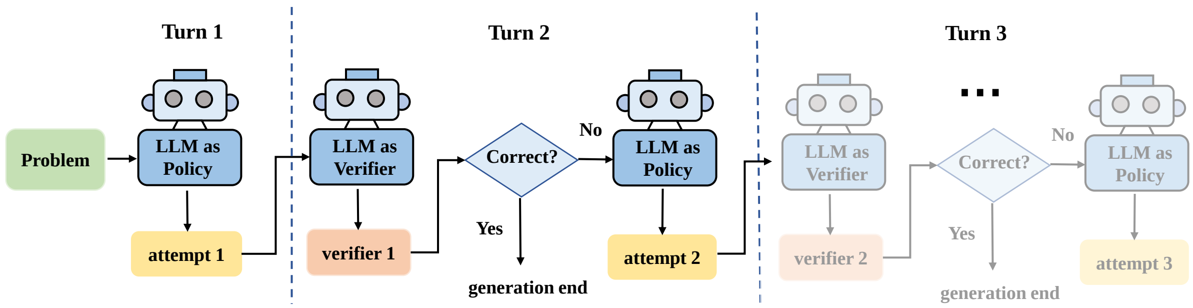


Figure 2 Overview of the Policy as Generative Verifier (PAG) framework. The LLM alternates between a policy role (generating solutions) and a generative verifier role (evaluating its own solutions) in a multi-turn process. This iterative refinement continues until self-verification is correct or a maximum number of turns is reached.

that simultaneously optimizes both the the policy and the generative verifier and enabling them to co-evolve and reinforce each other's capabilities throughout the training process.

Multi-Turn RL Recent studies have investigated multi-turn reinforcement learning (RL) for large language models (LLMs) from various perspectives, including preference-based [33, 47, 61], value-based [1, 60], and on-policy [13, 45] approaches. Multi-turn RL is emerging as a highly promising direction for developing LLM agents [27], though the field remains in its early stages.

3 Policy as Generative Verifier

This section introduces the Policy as Generative Verifier (PAG) framework, which enables large language models (LLMs) to perform multi-turn self-correction. As illustrated in Figure 2, PAG alternates the LLM between two distinct roles: a policy that proposes candidate solutions, and a verifier that critically evaluates the generated solutions through self-verification. This iterative process continues until the solution is verified as correct or a maximum number of turns is reached. We begin by formally outlining the framework and problem formulation.

3.1 Framework and Problem Formulation

Given an input problem $\mathbf{x}$, the LLM policy π_θ first generates an initial solution attempt $\hat{\mathbf{y}}_1$. In each subsequent turn t, the model produces a self-verification $\hat{\mathbf{v}}_t$ for the previous attempt $\hat{\mathbf{y}}_{t-1}$. If the verification indicates correctness (*i.e.*, $C(\hat{\mathbf{v}}_t, \hat{\mathbf{y}}_{t-1}, \mathbf{x}) = 1$), the process terminates. Otherwise, the LLM generates a revised attempt $\hat{\mathbf{y}}_t$, and the cycle repeats. The process continues until either the attempt is verified as correct or a predefined maximum number of turns $T_{\max}$ is reached. During training, both attempt and verification generations receive reward signals from a *external ground-truth verifier*. At inference time, the model must rely exclusively on its own self-verification, without any external feedback.

Formally, for each input $\mathbf{x}$, the model generates a trajectory $\tau_{\mathbf{x}} = (\mathbf{x}, \hat{\mathbf{y}}_1, \hat{\mathbf{v}}_2, \hat{\mathbf{y}}_2, \hat{\mathbf{v}}_3, \ldots)$ until termination. Let $N_A(\tau_{\mathbf{x}})$ and $N_V(\tau_{\mathbf{x}})$ denote the number of attempts and verifications in the trajectory, respectively. We define $\widehat{R}_y(\hat{\mathbf{y}}_i, \mathbf{x})$ as the reward for the i-th attempt and $\widehat{R}_v(\hat{\mathbf{v}}_j, \hat{\mathbf{y}}_{j-1}, \mathbf{x})$ as the reward for the j-th verification. Both rewards are binary: $\widehat{R}_y(\hat{\mathbf{y}}_i, \mathbf{x}) = 1$ if the attempt is correct and 0 otherwise, while $\widehat{R}_v(\hat{\mathbf{v}}_j, \hat{\mathbf{y}}_{j-1}, \mathbf{x}) = 1$ if the verification accurately identifies the correctness of the attempt and 0 otherwise. The overall training objective is formulated as:

$$\max_{\theta} \;\; \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbb{E}_{\tau_{\mathbf{x}}\sim\pi_\theta}\left[\sum_{i=1}^{N_A(\tau_{\mathbf{x}})} \widehat{R}_y(\hat{\mathbf{y}}_i, \mathbf{x}) + \sum_{j=1}^{N_V(\tau_{\mathbf{x}})} \widehat{R}_v(\hat{\mathbf{v}}_{j+1}, \hat{\mathbf{y}}_j, \mathbf{x})\right]\right] \tag{1}$$

This objective encourages the model to generate both correct answers and effective self-verifications, facilitating iterative refinement and early termination upon correctness. Due to limited computational resources, we set $T_{\max} = 2$ in all experiments unless otherwise specified.

A key innovation of our framework lies in the introduction of a selective revision mechanism, enabled by the generative verifier. Unlike prior approaches such as SCoRe [13], which always generate a second attempt regardless of the model's confidence, PAG only triggers a new attempt when its self-verification explicitly determines that the previous attempt is wrong. This selective revision mechanism ensures that the model focuses its efforts on self-correction only when necessary, rather than redundantly revising. In the context of multi-turn reinforcement learning, this design is particularly crucial: it directly addresses the model collapse issue observed in SCoRe, where repeatedly generating new attempts without discrimination causes subsequent responses to merely replicate the initial attempt with superficial modifications, leading to a collapse toward non-correcting behavior. By conditioning the revision process on the model's self-verification, PAG not only enhances self-correction performance, but also removes the need for the two-stage RL procedure required by SCoRe, resulting in a more streamlined and efficient training process.

3.2 Multi-turn Reinforcement Learning

Standard single-turn reinforcement learning (RL) algorithms can be seamlessly extended to multi-turn LLM generation by treating the entire sequence of turns as a single, concatenated trajectory. In this formulation, the model's outputs from each turn are concatenated together, with rewards assigned at the end of each corresponding segment. This design enables us to leverage existing RL algorithms—such as Proximal Policy Optimization (PPO) [31]—without requiring architectural modifications, providing a straightforward and effective approach for extending RL to multi-turn scenarios.

However, PAG introduces a unique challenge: the LLM alternates between acting as a policy and as a verifier, with each role entailing fundamentally different objectives. To address this, we adopt a *turn-independent* optimization strategy. In this approach, values or advantages from later turns are not propagated back to previous turns; instead, optimization is applied exclusively to the current turn. This is mathematically equivalent to setting the turn-level discount factor to zero. We empirically find this design to be crucial for achieving stable and effective training in the PAG framework.

To further enhance the learning dynamics, we introduce two training techniques. First, inspired by SCoRe [13], we incorporate a *bonus reward* for policy turns, defined as the improvement in accuracy between consecutive attempts: $b_y(\hat{\mathbf{y}}_t, \mathbf{x}) = \alpha(\widehat{R}_y(\hat{\mathbf{y}}_t, \mathbf{x}) - \widehat{R}_y(\hat{\mathbf{y}}_{t-1}, \mathbf{x}))$, where α is a positive scaling factor. This reward shaping mechanism explicitly incentivizes the model to correct previous mistakes by providing positive rewards when an incorrect answer is revised to a correct one, while imposing penalties when a correct answer is erroneously changed.

Second, we propose *RoleAdvNorm*, a novel advantage normalization technique tailored for multi-role training. Rather than following the standard PPO practice of normalizing advantages across the entire batch trajectory [3, 29, 49], we normalize advantages separately for policy and verifier roles within the multi-turn trajectory. This role-specific normalization prevents the mixing of advantages from fundamentally different tasks, leading to more stable gradient updates. Our empirical results demonstrate that combining RoleAdvNorm with reward shaping can enhance the model's self-correction capabilities.

4 Experiments

To verify the effectiveness of PAG, we conducted extensive experiments across 3 different base models on various datasets. All methods are trained using PPO with identical training hyperparameters. The complete set of hyperparameters is detailed in Appendix B. Unless otherwise specified, all our experiments are conducted with two training turns and two evaluation turns.

Base Models. We adopt Qwen2.5-1.5B-Instruct, Qwen2.5-7B-Instruct [51], and Llama3-8B-Instruct [7] as our base models. Our self-correction framework requires the underlying model to support multi-turn dialogue; therefore, we perform reinforcement learning based on existing instruction-tuned models. Unlike prior works [13, 48], we *do not* employ any additional SFT finetuning or warm-up training.

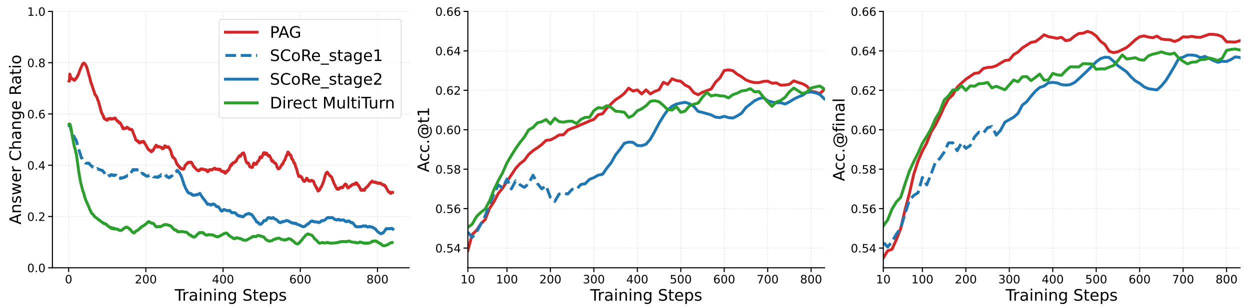


Figure 3 Training dynamics of PAG on Qwen2.5-1.5B-Instruct. Left: Answer change ratio quantifies model collapse as the proportion of responses where the second-turn answer differs from the first. Direct MultiTurn rapidly declines, indicating severe collapse. SCoRe partially alleviates this through two-stage RL, while PAG's selective revision mechanism effectively prevents collapse and achieves higher Acc.@t1 and Acc.@final; **Middle:** Acc.@t1; **Right:** Acc.@final.

Training and Evaluation Data. Our experiments primarily focus on mathematical reasoning tasks. For base models with relatively limited capabilities, such as Qwen2.5-1.5B-Instruct and Llama3-8B-Instruct, we use the MATH [9] training set, which contains 7.5K samples, and evaluate on the MATH500 [18]. For more capable models, such as Qwen2.5-7B-Instruct, we employ DAPO17K [53] for training, and evaluate on a broader set of mathematical benchmarks including MATH500, MinevaMATH [15], AIME2024, and AIME2025.

Baselines. Our baselines include: (1) **SingleTurn**, model only generate once during the RL training; (2) **Direct MultiTurn**, model directly generate two turns without self-verification; (3) **SCoRe** [13], builds on Direct Multiturn, but with two-stage RL training for better self-correction performance; We re-implemented SCoRe in our codebase, as the official implementation is not publicly available. (4) **Self-Rewarding** [48], we directly compare the performance with the reported results in their paper.

4.1 Policy Results

We first present the results for the model acting as a policy. For policy evaluation, we report **Acc.@t1** (accuracy of direct generation) and **Acc.@final** (accuracy after self-correction), with Acc.@final serving as the primary metric. In PAG, when the self-verifier identifies the first response as incorrect, the policy generates a second response, and Acc.@final reflects the accuracy of this revised attempt; otherwise, Acc.@final equals Acc.@t1. In contrast, baselines such as Direct MultiTurn and SCoRe invariably generate a second response, making Acc.@final equivalent to the second-turn accuracy.

PAG effectively mitigates model collapse. To quantitatively evaluate model collapse, we analyze the *answer change ratio*, defined as the proportion of instances where the model's second-turn answer differs from its first-turn answer. As illustrated in Figure 3, the answer change ratio for Direct MultiTurn exhibits a sharp decline during training, indicating rapid model collapse—a phenomenon also documented in SCoRe [13]. While SCoRe partially alleviates this issue through two-stage reinforcement learning, PAG fundamentally addresses model collapse by incorporating the selective revision mechanism: a second response is generated only when the first is explicitly identified as wrong. This selective revision strategy substantially enhances the answer change ratio, effectively prevents collapse, and yields both accelerated convergence and superior Acc.@final performance.

PAG achieves robust self-correction gains across different model scales and architectures. As shown in Table 1, PAG consistently achieves the highest self-correction performance (Acc.@final) on both Qwen2.5-1.5B-Instruct and Llama3-8B-Instruct. Specifically, PAG improves Acc.@final to 65.2% on Qwen2.5-1.5B-Instruct and 36.7% on Llama3-8B-Instruct, outperforming all baselines and significantly surpassing the respective base models. These results highlight the robustness and generality of PAG: it delivers strong gains across models of different parameter scales and architectures.

PAG achieves state-of-the-art self-correction on diverse benchmarks. As shown in Table 2, PAG achieves

Table 1 Performance on MATH500 with Qwen2.5-1.5B-Instruct and Llama3-8B-Instruct, evaluated using Avg@32. PAG not only achieves the best Acc.@final but also improves the Acc.@t1.

Method	Qwen2.5-1.5B-Instruct		Llama3-8B-Instruct	
	Acc.@t1	Acc.@final	Acc.@t1	Acc.@final
Base model	53.2%	52.4%	28.7%	26.6%
Single Turn	61.6%	62.4%	32.4%	32.4%
Direct MultiTurn	62.1%	64.0%	33.3%	35.4%
SCoRe [13]	61.9%	63.9%	32.0%	35.2%
Self-Rewarding [48]	-	-	25.0%	29.4%
PAG(Ours)	**62.2%**	**65.2%**	**35.1%**	**36.7%**

Table 2 Performance of PAG on diverse benchmarks with Qwen2.5-7B-Instruct, evaluated using Avg@32. Results are highlighted with underline and **bold** for the best Acc.@t1 and Acc.@final, respectively.

Model	MATH500	MinervaMath	AIME24	AIME25	Average
Qwen2.5-7B-Instruct					
Acc.@t1	76.0	35.9	11.1	7.6	32.7
Acc.@final	76.1	35.6	11.8	7.8	32.8
Single-Turn					
Acc.@t1	79.5	36.5	13.2	9.6	34.7
Acc.@final	80.2	37.1	14.1	10.6	35.5
Direct Multi-Turn					
Acc.@t1	80.0	35.3	15.8	<u>11.9</u>	35.8
Acc.@final	81.2	35.4	17.3	13.8	36.9
SCoRe					
Acc.@t1	79.5	36.7	13.1	10.0	34.8
Acc.@final	81.5	**37.7**	16.6	14.0	37.4
PAG(Ours)					
Acc.@t1	<u>80.3</u>	<u>37.0</u>	<u>16.6</u>	11.4	<u>36.3</u>
Acc.@final	**82.3**	37.2	**18.4**	**15.1**	**38.3**

the highest self-correction performance (Acc.@final) on Qwen2.5-7B-Instruct across all evaluated datasets. Specifically, PAG improves Acc.@final to 82.3% on MATH500 and achieves the best average performance (38.3%), consistently outperforming all baselines. These results further confirm the robustness and effectiveness of PAG for self-correction on diverse mathematical benchmarks.

4.2 Generative Verifier Results

In this subsection, we evaluate the performance of our model as a generative verifier. For generative verifier evaluation, we report three key metrics: **Correct Recall** (the proportion of correct answers that are correctly identified by the verifier), **Wrong Recall** (the proportion of incorrect answers that are correctly identified as wrong), and **Verifier Accuracy** (the overall accuracy of the verifier across all samples). Additionally, we report results for **self-verify BoN** (Best of N), where the model generates N candidate responses in parallel for each prompt and subsequently acts as a verifier to assess the correctness of each response. The final answer is selected as the response that receives the highest probability for the "correct" token.

PAG achieves high verifier accuracy on both self-generated and offline datasets. Table 3 and Figure 4 present the performance of the Generative Verifier on both self-generated responses from the MATH500 dataset and the offline RewardBench mathprm benchmark, respectively. The results demonstrate that baseline models (Qwen2.5-1.5B/7B-Instruct) exhibit limited self-verification capabilities. Notably, the SCoRe baseline,

Table 3 Performance on self-generated repsponses on MATH500. PAG improves the verifier accuracy by a large margin.

Method	Verifier Accuracy	Correct Recall	Wrong Recall
Qwen2.5-1.5B-Instruct	30.3	52.9	4.4
Qwen2.5-1.5B-Instruct + SCoRe	8.2	14.3	0.4
Qwen2.5-1.5B-Instruct + PAG	**81.7**	**91.2**	**65.8**
Qwen2.5-7B-Instruct	47.5	57.8	14.3
Qwen2.5-7B-Instruct + SCoRe	62.5	74.6	15.2
Qwen2.5-7B-Instruct + PAG	**90.7**	**94.2**	**76.4**

Model	**Score**
Qwen2.5-1.5B-Instruct	23.9
Qwen2.5-7B-Instruct	37.8
gemini-1.5-pro-0924	83.9*
Meta-Llama-3.1-70B-Instruct	76.4*
gpt-4-0125-preview	76.3*
Qwen2.5-1.5B-Instruct + PAG	78.5
Qwen2.5-7B-Instruct + PAG	86.6

Figure 4 Performance on RewardBench math-prm. Scores* are taken from RewardBench report.

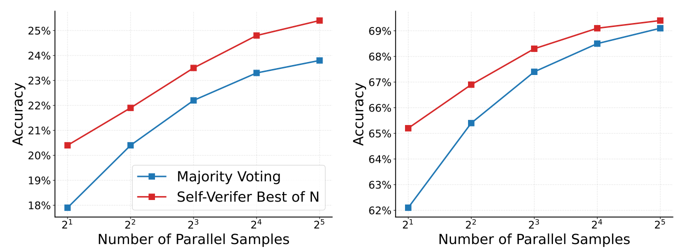


Figure 5 PAG self-verify BoN outperforms majority voting. Left: PAG with 7B model on AIME2024; **Right**: PAG with 1.5B model on MATH500.

which also involves answer revision, shows even poorer verifier performance than the base model for the 1.5B model as shown in Table 3. While SCoRe provides some improvement for the 7B model, its gains are modest compared to those achieved by PAG. This suggests that merely allowing answer revision, as SCoRe does, is not a consistently effective or effective strategy for developing strong self-verification capabilities. However, after incorporating the PAG framework, both models achieve substantial improvements across all metrics. Specifically, Verifier Accuracy on MATH500 increases by over 50 points for the 1.5B model and over 40 points for the 7B model. On RewardBench, the PAG-enhanced models not only outperform their respective baselines by large margins but also surpass several strong large-scale models, including Gemini [42], Llama-3.1 [7], and GPT-4 [2]. These findings highlight the effectiveness and strong generalization ability of the PAG framework in enhancing model self-verification, even when trained solely on policy data.

PAG self-verify BoN outperforms majority voting. We compare the performance of majority voting and self-verify BoN using the Qwen2.5-1.5B-Instruct model on MATH500 and the Qwen2.5-7B-Instruct model on AIME2024. As illustrated in Figure 5, PAG self-verify BoN consistently outperforms majority voting, achieving an average accuracy improvement of approximately 1% on MATH500 and 1.5% on AIME2024. This result is particularly noteworthy given that prior work [11] has demonstrated that self-verification typically underperforms compared to majority voting. Our findings indicate that this limitation no longer holds under the PAG framework, providing compelling evidence that the PAG training paradigm substantially enhances the model's self-verification capabilities.

4.3 Ablation Studies

We conduct ablation studies on several key components of our method: (1) The selective revision mechanism, which revises the initial attempt only when self-verification identifies the answer as incorrect—this represents the core design principle of PAG. (2) Turn-independent optimization, which is the fundamental design of our multi-turn RL algorithm. (3) RoleAdvNorm and reward shaping techniques within our multi-turn RL algorithm.

Selective revision is essential. We find that revising the initial policy attempt only when self-verification

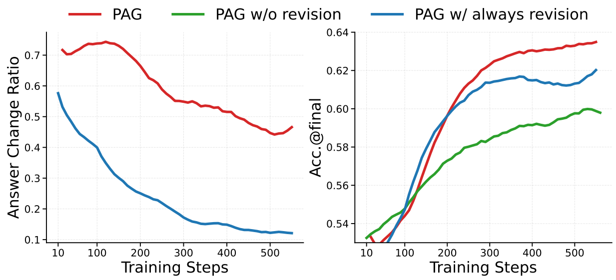


Figure 6 Ablation of selective revision. *PAG w/ always revision* rapidly collapses, while *PAG w/o revision* yields lower Acc.@final.

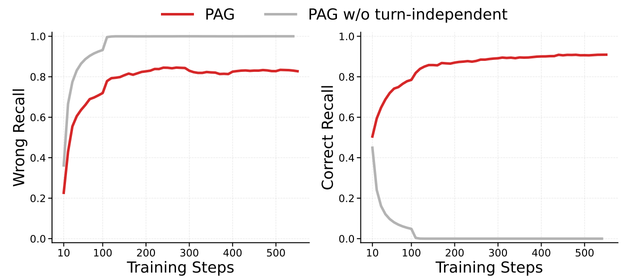


Figure 7 Ablation of turn-independent optimization. The verifier capability of *PAG w/o turn-independent* collapses, consistently evaluating all answers as wrong.

judges the previous answer as wrong is crucial for effective self-correction. To validate this, we conduct ablation studies with two variants: (1) *PAG w/o revision*, which generates only a single response and a single verifier output (*i.e.*, no revision), and (2) *PAG w/ always revision*, which always revises the initial attempt regardless of the self-verification outcome. As shown in Figure 6, PAG significantly outperforms both *PAG w/o revision* and *PAG w/ always revision*. The superiority of *PAG* over *PAG w/o revision* is intuitive, as the latter lacks any self-correction mechanism. Notably, *PAG w/ always revision* essentially degenerates into Direct MultiTurn, and we observe that its answer change ratio decreases rapidly, which fundamentally constrains its self-correction capability.

Turn-independent optimization prevents verifier collapse. We compare our turn-independent optimization approach with a variant that propagates advantages across turns (*i.e.*, "w/o turn-independent optimization"). As shown in Figure 7, removing turn-independent optimization causes the generative verifier to collapse: the model consistently outputs "The answer is wrong", with correct recall dropping to zero and wrong recall rising to one. The fundamental issue is that the verifier should only be rewarded for making accurate judgments. However, when its advantage is contaminated by rewards from subsequent policy turns, the verifier can receive positive rewards even when making incorrect judgments. This leads to reward hacking, where the model is incentivized to always trigger additional turns, regardless of the actual correctness of the answer. Turn-independent optimization decouples verifier and policy training, preventing such interference and ensuring stable verifier learning.

Table 4 Ablations of RoleAdvNorm and reward shaping. Combining RoleAdvNorm and reward shaping yields the best self-correction performance, while using either alone brings no improvement.

RoleAdvNorm	Reward Shaping	Acc.@t1	Acc.@final
		62.0%	64.7%
	✓	61.5%	64.4%
✓		61.8%	64.7%
✓	✓	**62.2%**	**65.2%**

The combination of RoleAdvNorm and reward shaping provides moderate improvements in self-correction performance. We conduct ablation studies to examine the effects of reward shaping and task normalization (RoleAdvNorm) on Qwen2.5-1.5B-Instruct, evaluating on MATH500. As shown in Table 4, PAG achieves the best results when both reward shaping and RoleAdvNorm are applied together. In contrast, applying either reward shaping or RoleAdvNorm alone does not yield improvements. This can be attributed to the following: reward shaping is designed to enhance the model's performance in the second policy turn. However, because a verifier step is interleaved between the two policy turns, the verifier's advantage is also included in the advantage normalization, which may diminish the effect of reward shaping. When RoleAdvNorm is introduced, the verifier's advantage is excluded from the normalization process across the two policy turns, thereby allowing reward shaping to take full effect and improving overall performance.

4.4 Sequential Self-Correction Sampling vs Parallel Sampling

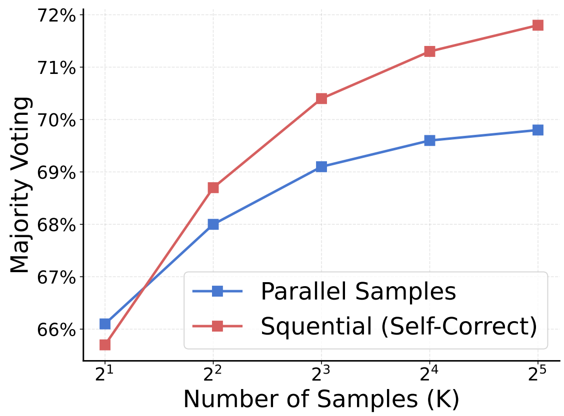


Figure 8 Sequential Sampling vs Parallel Sampling

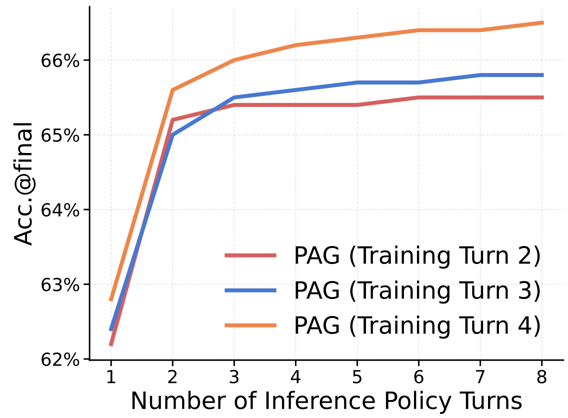


Figure 9 Scaling Turns of PAG.

We compare two paradigms for inference-time scaling: sequential self-correction sampling and parallel sampling. Sequential self-correction sampling generates K solutions in parallel, then applies one round of self-correction to each solution, while parallel sampling directly samples $2K$ solutions in parallel. We employ majority voting as the aggregation metric for final answers, conducting experiments on Qwen2.5-1.5B-Instruct and evaluating performance on MATH500. As illustrated in Figure 8, sequential self-correction sampling demonstrates substantially superior computational efficiency compared to parallel sampling. Notably, sequential sampling with self-correction at $K = 8$ achieves performance superior to parallel sampling at $K = 32$, yielding a remarkable $4\times$ improvement in computational efficiency. These results underscore the significant practical advantages of integrating sequential self-correction into the sampling framework, which aligns with recent findings in the literature [13, 39].

4.5 Scaling Training and Inference Turns of PAG

To better understand the scalability of our approach, we investigate two critical factors that influence model performance: the number of training policy turns and the number of inference policy turns. We extend the policy turns during inference to 8 and scale the training policy turns up to 4. Figure 9 presents the results of this analysis on the model trained using PAG with Qwen2.5-1.5B-Instruct and evaluated on MATH500.

Our findings reveal that scaling to more inference turns continues to improve accuracy, though the marginal gains diminish significantly compared to the substantial improvement observed from the first to the second policy turn. Importantly, scaling training turns from 2 to 4 enhances these marginal improvements, with the final accuracy at the eighth policy attempt increasing proportionally with the number of training turns. Specifically, increasing the training turns from two to three yields a 0.3% gain in Acc.@final with 8 policy attempts, while further scaling from two to four training turns results in a 1.0% improvement in accuracy. We leave more effective approaches for scaling training turns as future work.

5 Conclusion

We introduce PAG (Policy as Generative Verifier), a novel framework efficiently training a single LLM to serve dual roles as generator and verifier for multi-turn self-correction. PAG streamlines training by circumventing complex warm-up stages, directly fine-tuning instruction-following models, and mitigating model collapse. Extensive experiments on mathematical reasoning benchmarks show PAG significantly outperforms existing methods in both generation accuracy and self-correction capabilities. Moreover, PAG exhibits strong self-verification: its self-selected best-of-N sampling achieves superior performance over self-consistency approaches. One limitation of PAG is its reliance on external ground-truth verifiers to provide reward signals during training, which restricts its applicability to tasks where such supervision is unavailable.

References

[1] Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models, 2023.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[3] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=nIAxjsniDzg.

[4] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128, 2023.

[5] Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, Hanghang Tong, and Heng Ji. Rm-r1: Reward modeling as reasoning, 2025.

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

[7] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

[8] 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.

[9] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

[10] Audrey Huang, Adam Block, Dylan J Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan T. Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=WJaUkwci9o.

[11] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. arXiv preprint arXiv:2310.01798, 2023.

[12] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. arXiv preprint arXiv:2403.07974, 2024.

[13] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. arXiv preprint arXiv:2409.12917, 2024.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

[15] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. Advances in Neural Information Processing Systems, 35:3843–3857, 2022.

[16] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161, 2023.

[17] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. Science, 378 (6624):1092–1097, 2022.

[18] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In The Twelfth International Conference on Learning Representations, 2023.

[19] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. arXiv preprint arXiv:2308.03688, 2023.

[20] Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling, 2025.

[21] Ziming Luo, Zonglin Yang, Zexin Xu, Wei Yang, and Xinya Du. Llm4sr: A survey on large language models for scientific research. arXiv preprint arXiv:2501.04306, 2025.

[22] Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. S2r: Teaching llms to self-verify and self-correct via reinforcement learning. arXiv preprint arXiv:2502.12853, 2025.

[23] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. arXiv preprint arXiv:2303.17651, 2023.

[24] Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models. arXiv preprint arXiv:2410.12832, 2024.

[25] OpenAI. O1 by openai. https://openai.com/o1/, 2024.

[26] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. arXiv preprint arXiv:2501.14249, 2025.

[27] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. arXiv preprint arXiv:2501.12326, 2025.

[28] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. arXiv preprint arXiv:2407.18219, 2024.

[29] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[30] William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. arXiv preprint arXiv:2206.05802, 2022.

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[32] ByteDance Seed, Yufeng Yuan, Yu Yue, Mingxuan Wang, Xiaochen Zuo, Jiaze Chen, Lin Yan, Wenyuan Xu, Chi Zhang, Xin Liu, et al. Seed-thinking-v1. 5: Advancing superb reasoning models with reinforcement learning. arXiv preprint arXiv:2504.13914, 2025.

[33] Lior Shani, Aviv Rosenberg, Asaf Cassel, Oran Lang, Daniele Calandriello, Avital Zipori, Hila Noga, Orgad Keller, Bilal Piot, Idan Szpektor, et al. Multi-turn reinforcement learning from preference human feedback. arXiv preprint arXiv:2405.14655, 2024.

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

[35] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. arXiv preprint arXiv: 2409.19256, 2024.

[36] Wenlei Shi and Xing Jin. Heimdall: test-time scaling on the generative verification, 2025.

[37] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. arXiv preprint arXiv:2303.11366, 2023.

[38] Sina Shool, Sara Adimi, Reza Saboori Amleshi, Ehsan Bitaraf, Reza Golpira, and Mahmood Tara. A systematic review of large language model (llm) evaluations in clinical medicine. BMC Medical Informatics and Decision Making, 25(1):117, 2025.

[39] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

[40] Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, and Udaya Ghai. Mind the gap: Examining the self-improvement capabilities of large language models. arXiv preprint arXiv:2412.02674, 2024.

[41] Hao Sun, Yunyi Shen, and Jean-Francois Ton. Rethinking reward modeling in preference-based large language model alignment. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=rfdblE10qm.

[42] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.

[43] Gladys Tyen, Hassan Mansoor, Victor Cărbune, Yuanzhu Peter Chen, and Tony Mak. Llms cannot find reasoning errors, but can correct them given the error location. In Findings of the Association for Computational Linguistics ACL 2024, pages 13894–13908, 2024.

[44] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171, 2022.

[45] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. arXiv preprint arXiv:2504.20073, 2025.

[46] Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge. arXiv preprint arXiv:2407.19594, 2024.

[47] Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. arXiv preprint arXiv:2409.02392, 2024.

[48] Wei Xiong, Hanning Zhang, Chenlu Ye, Lichang Chen, Nan Jiang, and Tong Zhang. Self-rewarding correction for mathematical reasoning. arXiv preprint arXiv:2502.19613, 2025.

[49] Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is DPO superior to PPO for LLM alignment? a comprehensive study. In Forty-first International Conference on Machine Learning, 2024. URL https://openreview.net/forum?id=6XH8R7YrSk.

[50] Wenyuan Xu, Xiaochen Zuo, Chao Xin, Yu Yue, Lin Yan, and Yonghui Wu. A unified pairwise framework for rlhf: Bridging generative reward modeling and policy optimization. arXiv preprint arXiv:2504.04950, 2025.

[51] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. arXiv preprint arXiv:2412.15115, 2024.

[52] Seonghyeon Ye, Yongrae Jo, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, and Minjoon Seo. Selfee: Iterative self-revising llm empowered by self-feedback generation. Blog post, 2023.

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

[54] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models, 2025. URL https://arxiv.org/abs/2401.10020.

[55] Yufeng Yuan, Yu Yue, Ruofei Zhu, Tiantian Fan, and Lin Yan. What's behind ppo's collapse in long-cot? value optimization holds the secret. arXiv preprint arXiv:2503.01491, 2025.

[56] Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Ru Zhang, Xin Liu, Mingxuan Wang, Yonghui Wu, and Lin Yan. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks, 2025. URL https://arxiv.org/abs/2504.05118.

[57] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. Advances in Neural Information Processing Systems, 35:15476–15488, 2022.

[58] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=Ccwp4tFEtE.

[59] Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning, 2025. URL https://arxiv.org/abs/2501.07301.

[60] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. arXiv preprint arXiv:2402.19446, 2024.

[61] Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. arXiv preprint arXiv:2503.15478, 2025.

Appendix

A Additional Experiments

A.1 Comparison with Direct Multi-Turn Training Using Three Turns

Method	Qwen2.5-1.5B-Instruct	
	Acc.@t1	Acc.@final
Direct Multi-Turn (2 Turns)	62.1%	64.0%
Direct Multi-Turn (3 Turns)	62.3%	64.5%
PAG(Ours)	62.2%	**65.2%**

Table 5 Comparison of PAG and Direct Multi-Turn (2 and 3 turns) on MATH500 using Qwen2.5-1.5B-Instruct. Increasing the number of training turns in Direct Multi-Turn improves Acc.@final, but PAG still achieves the best performance.

We note that PAG's two-turn process consists of two policy generations and one verifier generation, resulting in three outputs from the LLM. In contrast, baseline Direct Multi-Turn methods typically involve only two policy generations. To ensure a fair comparison, we train the Direct Multi-Turn baseline with three turns and compare it to PAG's two-turn setting. It is important to highlight that, due to PAG's selective revision mechanism, not every problem requires three outputs, whereas Direct Multi-Turn always generates three outputs for every problem, making PAG more token-efficient. As shown in Table 5, increasing the number of training turns for Direct Multi-Turn from two to three leads to a higher final accuracy. However, PAG still achieves superior final accuracy, demonstrating its effectiveness both in terms of token efficiency and self-correction performance. We do not compare with SCoRe [13], as it is specifically designed for two-turn settings.

A.2 Scaling the Number of Policy Attempts: Impact on Iterative Self-Correction

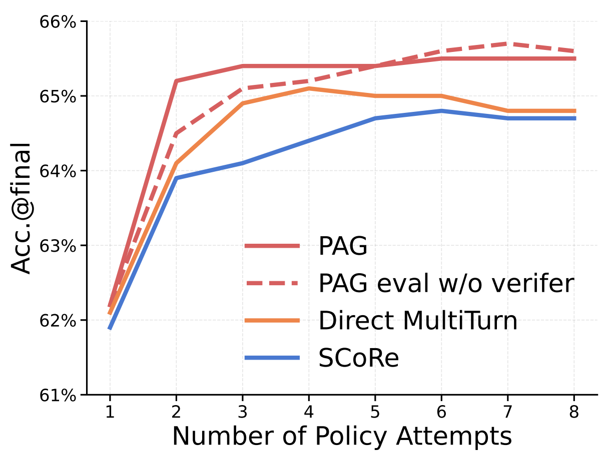


Figure 10 Performance comparison of PAG, Direct Multi-Turn, and SCoRe across 8 policy attempts, despite being trained with only two policy attempts.

We investigate how various models perform when required to iteratively self-correct over multiple attempts, even though they were trained using only two policy attempts. We compare the performance of PAG, Direct Multi-Turn, and SCoRe across different numbers of inference turns. We also introduce a variant called "PAG eval w/o verifier," where we evaluate the PAG-trained model with the verifier role removed, making the revision process identical to that used by Direct Multi-Turn and SCoRe.

As shown in Figure 10, increasing the number of policy attempts leads to performance improvements, although the gains are relatively modest. PAG consistently outperforms both Direct Multi-Turn and SCoRe across all settings. Notably, as the number of policy attempts increases, the performance of PAG evaluated without the verifier (PAG eval w/o verifier) approaches that of the full PAG framework. This suggests that training with PAG effectively enhances the model's self-correction capability, making the presence of the verifier during evaluation less critical when more turns are allowed.

A.3 Ablation of Standalone Generative Verifier: The Importance of Online Training

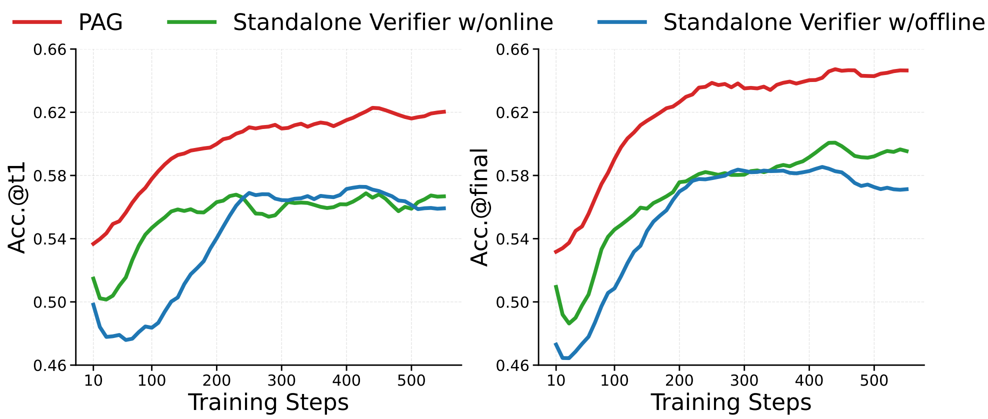


Figure 11 Policy Performance of Standalone Generative Verifier on MATH500

Model	Score
Qwen2.5-1.5B-Instruct	23.9
Qwen2.5-1.5B-Instruct + PAG	78.5
Qwen2.5-1.5B-Instruct + Standalone Verifier (online)	78.1
Qwen2.5-1.5B-Instruct + Standalone Verifier (offline)	72.0

Table 6 Verifier Performance Standalone Generative Verifier on RewardBench mathprm.

In PAG, the verifier is co-trained with the policy using on-policy data. This ablation study investigates the implications of training a generative verifier independently via reinforcement learning. We explore two strategies for generating the verifier's training data: (1) **Standalone Verifier (offline)**: The Qwen1.5B-Instruct model initially generates four responses per MATH question. These responses, serving as prompts for the verifier's tasks, are labeled as correct or incorrect to constitute the verifier's training dataset. (2) **Standalone Verifier (online)**: Following each update to the policy model, new responses are generated for the same set of questions. These fresh responses then become the updated prompts for verifier training, and the verifier is trained on this dynamically generated data. This online process is analogous to two generation rounds in PAG but omits the revision step and does not leverage first-round outputs for policy optimization.

To evaluate policy performance, we employ the PAG framework. As illustrated in Figure 11, we report the policy results. While the standalone verifier does not surpass the full PAG framework, we find that training the verifier alone still enhances policy performance, as evidenced by improvements in both Acc.@t1 and Acc.@final. Notably, the Standalone Verifier w/online achieves a higher Acc.@final than its offline counterpart, suggesting that online training of the verifier is more effective by mitigating distribution shift.

To evaluate the verifier, we use the RewardBench prm dataset, with results presented in Table 11. We observe that the standalone verifier (offline) exhibits weaker verification capability compared to the standalone verifier (online), as evidenced by a 6.1-point lower score on RewardBench. This indicates that online training leads to a more effective verifier than offline training. Notably, the verification ability of the online-trained standalone verifier is already comparable to that of the full PAG verifier. PAG does not significantly improve the verifier's standalone capability; rather, it leverages the verifier's strength to enhance the overall efficiency and effectiveness of self-correction in the model.

A.4 Ablation of Multi-Turn Reward: The Importance of Per-Turn Rewards

In the standard multi-turn training paradigm of PAG, both policy and verifier outputs at each turn are assigned ground-truth rewards. In this section, we systematically investigate the effect of providing rewards only at specific turns, rather than at every turn. We consider two ablation variants: (1) **PAG w/ final reward only**: Only the final output—regardless of whether it is generated by the policy or the verifier—receives a reward. For this setting, the turn-level discount factor is set to 1. (2) **PAG w/o first policy reward**: The policy output at the first turn does not receive a reward; instead, the verifier output at the second turn is rewarded, and if a second policy output is generated, it also receives a reward. In this case, a turn discount of 1 is applied between the first policy output and the second verifier output, and training is truncated at the third turn.

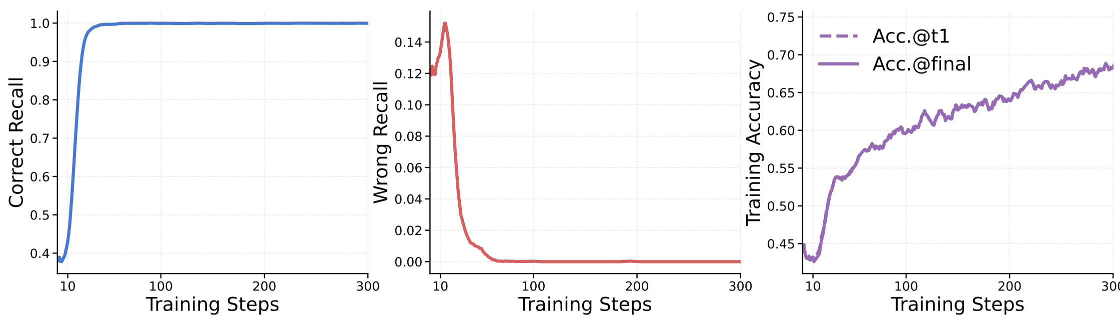


Figure 12 Ablation of PAG with final reward only.

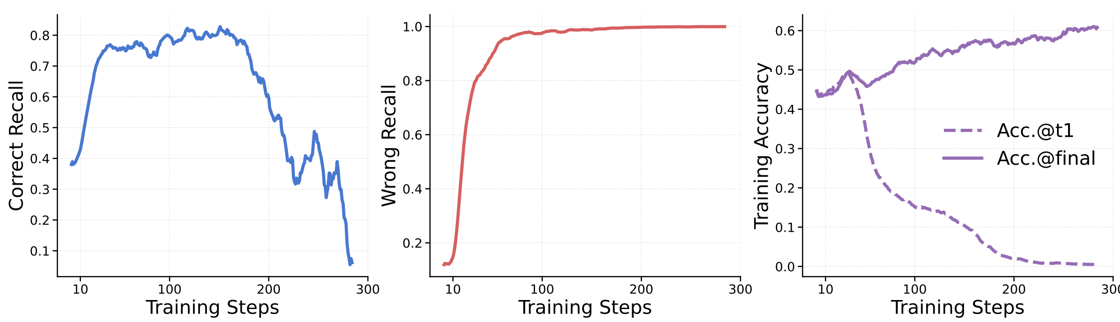


Figure 13 Ablation of PAG without first policy reward.

Figure 12 shows the results for PAG when only the final output is rewarded (PAG w/ final reward only). We observe that the verifier collapses to always output "The answer is correct," as indicated by Correct Recall converging to 1 and Wrong Recall dropping to 0. In this scenario, the model is incentivized to maximize Acc.@t1, since this directly increases the reward by encouraging the verifier to always confirm the answer as correct. This is further evidenced by the continuous increase of Acc.@t1 on the training set. As a result, Acc.@final becomes identical to Acc.@t1, since no revision occurs when the verifier always outputs "The answer is correct."

Figure 13 presents the results for PAG without the first policy reward (PAG w/o first policy reward). In this setting, the policy collapses to almost always output "The answer is wrong," with Wrong Recall rapidly decreasing to 0. This phenomenon occurs because, without a reward for the first policy output, the model is incentivized to produce arbitrary answers in the first round, prompting the verifier to label them as wrong and thus receive the verifier's reward. Subsequent revision then improves the answer, leading to a normal increase in Acc.@final.

In summary, our findings demonstrate that removing the reward at any turn leads to a collapse in either policy or verifier performance. Therefore, providing rewards at every turn is crucial for stable and effective training in PAG.

B Implementation Details

Our implementation is based on the VeRL framework [35], and we utilize vLLM 0.8.2 [14] for our experiments. During reinforcement learning training, we do not apply KL regularization. Table 7 summarizes the hyper-parameters employed in our experiments. For reward shaping in both PAG and SCoRe, we use $\alpha = 1$; we have also experimented with $\alpha = 5$ and $\alpha = 10$, but observed no performance improvements. The maximum response length per turn in our experiments varies by model: 4096 tokens for Qwen2.5-7B-Instruct, 2048 tokens for Qwen2.5-1.5B-Instruct, and 2048 tokens for Llama-3-8B-Instruct. For the 1.5B model, we utilize 8×H800 GPUs with a training duration of less than 24 hours. For the 7B model, we employ 32xH800 GPUs with a training duration of less than 40 hours.

For evaluation purposes, we select checkpoints with the highest validation performance. We generate 32 responses per prompt with temperature set to 0.6 , top-k to -1 and top-p to 0.95 [35].

Table 7 Hyperparameters used in our experiments During RL Training.

Hyperparameter	Value
Temperature	1.0
Top-p	1.0
Actor learning rate	1×10^{-6}
Critic learning rate	2×10^{-6}
Training prompt batch size	512
Responses per prompt	4
PPO epochs	1
PPO mini-batch size	512
PPO clip_high	0.28
PPO clip_low	0.2
Entropy loss coefficient	0
KL coefficient	0

C Limitations and Future Work

Beyond extending PAG to settings without golden reward models for general tasks, another critical avenue for future research involves investigating the scalability of PAG to larger models (e.g., 32B/70B parameters) and its integration with advanced reasoning techniques, such as long chain-of-thought [8, 25, 32]. We defer these promising directions to future work.

Finally, regarding the choice of reinforcement learning algorithms, our current work employs PPO for multi-turn training. We have also experimented with the REINFORCE algorithm but found it to exhibit instability in our experimental setting. Other promising algorithms, such as GRPO [34], DAPO [53], and VAPO [55, 56], remain unexplored and constitute compelling opportunities for future investigation.

D Examples of PAG Self-Correction

MATH EXAMPLE 1

System:. Please reason step by step, and put your final answer within $\boxed{}$.

User:. A regular hexagon can be divided into six equilateral triangles. If the perimeter of one of the triangles is 21 inches, what is the perimeter, in inches, of the regular hexagon?

Assistant:. To determine the perimeter of the regular hexagon, we start by analyzing the properties of the equilateral triangles that make up the hexagon. Since the hexagon is divided into six equilateral triangles, each of these triangles has a side length equal to the side length of the hexagon.
We know that the perimeter of one of these equilateral triangles is 21 inches. Since an equilateral triangle has three equal sides, the side length of each equilateral triangle is:

$$\frac{21 \text{ inches}}{3} = 7 \text{ inches}$$

Each side of the equilateral triangles is therefore 7 inches. Since the hexagon is composed of six such equilateral triangles, the perimeter of the hexagon is six times the side length of one triangle:

$$6 \times 7 \text{ inches} = 42 \text{ inches}$$

Thus, the perimeter of the regular hexagon is $\boxed{42}$ inches.

User: Check the math solution step-by-step. If you find a mistake: state the wrong step, explain why it's wrong, and end your response with 'The answer is wrong'. If all steps are correct, end your response with 'The answer is correct'.

Assistant: Let's go through the problem step by step to find the perimeter of the regular hexagon.
First, we know that a regular hexagon can be divided into six equilateral triangles. If the perimeter of one of these equilateral triangles is 21 inches, we need to determine the side length of one of these equilateral triangles.
Since each equilateral triangle has three sides of equal length, the side length s of one of these triangles is:

$$s = \frac{21 \text{ inches}}{3} = 7 \text{ inches}$$

Now, since a regular hexagon is composed of six such equilateral triangles, the perimeter of the hexagon is six times the side length of one equilateral triangle. So the perimeter P of the hexagon is:

$$P = 6 \times s = 6 \times 7 = 42 \text{ inches}$$

Thus, the perimeter of the regular hexagon is 42 inches.
The answer is correct.

MATH EXAMPLE 2

System: Please reason step by step, and put your final answer within $\boxed{}$.

User: For how many two-digit primes is the sum of the digits equal to 8?

Assistant: To determine how many two-digit prime numbers have digits that sum to 8, we first need to identify all two-digit numbers whose digits sum to 8. A two-digit number can be represented as $10a + b$ where a and b are its digits, and a ranges from 1 to 9, while b ranges from 0 to 9. We need $a + b = 8$.
Let's find all possible values of a and b:
- If $a = 1$, then $b = 7$, giving the number 17.
- If $a = 2$, then $b = 6$, giving the number 26.
- If $a = 3$, then $b = 5$, giving the number 35.
- If $a = 4$, then $b = 4$, giving the number 44.
- If $a = 5$, then $b = 3$, giving the number 53.
- If $a = 6$, then $b = 2$, giving the number 62.
- If $a = 7$, then $b = 1$, giving the number 71.
- If $a = 8$, then $b = 0$, giving the number 80, but 80 is not a two-digit number.

So the two-digit numbers whose digits sum to 8 are: 17, 26, 35, 44, 53, 62, 71, and 80. However, we only need to consider the two-digit numbers, so we exclude 80. Thus, the valid two-digit numbers are: 17, 26, 35, 44, 53, 62, and 71.
Next, we need to check which of these numbers are prime:
- 17 is a prime number.
- 26 is not a prime number (divisible by 2 and 13).
- 35 is not a prime number (divisible by 5 and 7).
- 44 is not a prime number (divisible by 2 and 11).
- 53 is a prime number.
- 62 is not a prime number (divisible by 2 and 31).
- 71 is a prime number.

Thus, the prime numbers among the list are 17, 53, and 71. Therefore, the two-digit prime numbers whose digits sum to 8 are 17, 53, and 71.
Counting these, we find there are 3 such prime numbers. Therefore, the number of two-digit primes whose digits sum to 8 is $\boxed{3}$.

User: Check the math solution step-by-step. If you find a mistake: state the wrong step, explain why it's wrong, and end your response with 'The answer is wrong'. If all steps are correct, end your response with 'The answer is correct'.

Assistant: Let's first identify all two-digit numbers whose digits sum to 8. We need to find all numbers of the form $10a + b$ where a and b are digits (*i.e.*, a ranges from 1 to 9 and b ranges from 0 to 9), and $a + b = 8$.
We can list these numbers as follows: - If $a = 1$, then $b = 7$ giving the number 17.
- If $a = 2$, then $b = 6$ giving the number 26.
- If $a = 3$, then $b = 5$ giving the number 35.
- If $a = 4$, then $b = 4$ giving the number 44.
- If $a = 5$, then $b = 3$ giving the number 53.
- If $a = 6$, then $b = 2$ giving the number 62.
- If $a = 7$, then $b = 1$ giving the number 71.
- If $a = 8$, then $b = 0$ giving the number 80, but 80 is not a two-digit number.

So the two-digit numbers whose digits sum to 8 are: 17, 26, 35, 44, 53, 62, and 71. Now, we need to check which of these numbers are prime:
- 17 is a prime number.
- 26 is not a prime number (divisible by 2 and 13).
- 35 is not a prime number (divisible by 5 and 7).
- 44 is not a prime number (divisible by 2 and 11).
- 53 is a prime number.
- 62 is not a prime number (divisible by 2 and 31).
- 71 is a prime number.

The prime numbers from the list are 17, 53, and 71. Thus, the two-digit prime numbers whose digits sum to 8 are 17, 53, and 71. Counting these, we have 3 such numbers.
Therefore, the number of two-digit primes whose digits sum to 8 is 3.
The answer is correct.

MATH EXAMPLE 3

System: Please reason step by step, and put your final answer within $\boxed{}$.

User: What is the remainder when $129^{34} + 96^{38}$ is divided by 11?

Assistant: To find the remainder when $129^{34} + 96^{38}$ is divided by 11, we start by simplifying the numbers modulo 11. First, we reduce $129 \mod 11$:

$$129 \div 11 = 11 \text{ remainder } 8 \Longrightarrow 129 \equiv 8 \pmod{11}$$

Thus, $129^{34} \equiv 8^{34} \pmod{11}$.
Next, we reduce $96 \mod 11$:

$$96 \div 11 = 8 \text{ remainder } 8 \Longrightarrow 96 \equiv 8 \pmod{11}$$

Thus, $96^{38} \equiv 8^{38} \pmod{11}$.
Now we need to find the remainders of $8^{34} \mod 11$ and $8^{38} \mod 11$.
By Fermat's Little Theorem, since 11 is a prime number, we know that for any integer a such that a is not divisible by 11, $a^{10} \equiv 1 \pmod{11}$. Thus, we can reduce the exponents modulo 10 because the exponent modulo 10 will give the same result for the remainder when divided by 11.
First, let's simplify $8^{34} \mod 11$:

$$34 \div 10 = 3 \text{ remainder } 4 \Longrightarrow 34 \equiv 4 \pmod{10}$$

So, $8^{34} \equiv 8^4 \pmod{11}$.
Now, let's calculate $8^4 \mod 11$:

$$8^2 = 64 \equiv 10 \pmod{11} \quad (\text{since } 64 - 5 \times 11 = 64 - 55 = 9)$$

$$8^4 = (8^2)^2 = 10^2 = 100 \equiv 100 - 9 \times 11 = 100 - 99 = 1 \pmod{11}$$

So, $8^4 \equiv 1 \pmod{11}$, therefore $8^{34} \equiv 1 \pmod{11}$.
Next, let's simplify $8^{38} \mod 11$:

$$38 \div 10 = 3 \text{ remainder } 8 \Longrightarrow 38 \equiv 8 \pmod{10} \Longrightarrow 38 \equiv 8 \pmod{10}$$

So, $8^{38} \equiv 8^8 \pmod{11}$.
Now, let's calculate $8^8 \mod 11$:

$$8^2 = 64 \equiv 9 \pmod{11} \quad (\text{since } 64 - 5 \times 11 = 64 - 55 = 9)$$

$$8^4 = (8^2)^2 = 9^2 = 81 \equiv 4 \pmod{11} \quad (\text{since } 81 - 7 \times 11 = 81 - 77 = 4)$$

$$8^8 = (8^4)^2 = 4^2 = 16 \equiv 5 \pmod{11} \quad (\text{since } 16 - 1 \times 11 = 16 - 11 = 5)$$

So, $8^8 \equiv 5 \pmod{11}$.
Now we add the two results:

$$8^{34} + 8^{38} \equiv 1 + 5 \equiv 6 \pmod{11}$$

Thus, the remainder when $129^{34} + 96^{38}$ is divided by 11 is $\boxed{6}$.

User: Check the math solution step-by-step. If you find a mistake: state the wrong step, explain why it's wrong, and end your response with 'The answer is wrong'. If all steps are correct, end your response with 'The answer is correct'.

Assistant: Let's break it down step by step:
First, we need to find the remainders of 129^{34} and 96^{38} modulo 11.
1. Simplify $129 \mod 11$:

$$129 \div 11 = 11 \text{ remainder } 8 \Rightarrow 129 \equiv 8 \mod 11$$

So, $129^{34} \equiv 8^{34} \mod 11$.
2. Simplify $96 \mod 11$:

$$96 \div 11 = 8 \text{ remainder } 8 \Rightarrow 96 \equiv 8 \mod 11$$

So, $96^{38} \equiv 8^{38} \mod 11$.
Now we need to find $8^{34} \mod 11$ and $8^{38} \mod 11$.
Using Fermat's Little Theorem which states that $a^{p-1} \equiv 1 \mod p$ for a prime p and a not divisible by p, we know that $a^{p-1} \equiv 1 \mod p$. Here $p = 11$, so $a^{10} \equiv 1 \mod 11$ for any a not divisible by 11. So we can reduce the exponents modulo 10.
For $8^{34} \mod 11$:

$$34 \mod 10 = 4 \Rightarrow 8^4 \mod 11$$

Now we need to calculate $8^4 \mod 11$:

$$8^2 = 64 \equiv 64 - 5 \times 11 = 64 - 55 = 9 \mod 11$$

$$8^4 = (8^2)^2 = 9^2 = 81 \equiv 81 - 7 \times 11 = 81 - 77 = 4 \mod 11$$

So, $8^4 \equiv 4 \mod 11$.
For $8^{38} \mod 11$:

$$38 \mod 10 = 8 \Rightarrow 8^8 \mod 11$$

Now we need to calculate $8^8 \mod 11$:

$$8^2 = 64 \equiv 64 - 5 \times 11 = 64 - 55 = 9 \mod 11$$

$$8^4 = (8^2)^2 = 9^2 = 81 \equiv 81 - 7 \times 11 = 81 - 77 = 4 \mod 11$$

$$8^8 = (8^4)^2 = 4^2 = 16 \equiv 16 - 11 = 5 \mod 11$$

So, $8^8 \equiv 5 \mod 11$.
Now, adding these results:

$$8^{34} + 8^{38} \equiv 4 + 5 \equiv 9 \mod 11$$

Therefore, the remainder when $129^{34} + 96^{38}$ is divided by 11 is 9.
However, it seems there was an error in my previous calculation. Let's redo the calculations properly.
For $8^{34} \mod 11$:

$$8^{10} \equiv 1 \mod 11$$

So, $8^{34} = 8^{10 \cdot 3 + 4} = (8^{10})^3 \cdot 8^4 \equiv 1^3 \cdot 8^4 \equiv 8^4 \mod 11$. Now we calculate $8^4 \mod 11$:

$$8^2 = 64 \equiv 64 - 5 \cdot 11 = 64 - 55 = 9 \mod 11$$

$$8^4 = (8^2)^2 = 9^2 = 81 \equiv 81 - 7 \cdot 11 = 81 - 77 = 4 \mod 11$$

So, $8^4 \equiv 4 \mod 11$.
For $8^{38} \mod 11$:

$$8^{38} = 8^{10 \cdot 3 + 8} = (8^{10})^3 \cdot 8^8 \equiv 1^3 \cdot 8^8 \equiv 8^8 \mod 11$$

We already calculated $8^8 \equiv 5 \mod 11$.
Now adding them:

$$8^{34} + 8^{38} \equiv 4 + 5 \equiv 9 \mod 11$$

Thus, the remainder is 9.
The answer is wrong.

User: You indicated that your previous answer was wrong. Please provide the correct solution to the math problem.

Assistant: Let's break down the problem step by step.
We need to find the remainder when $129^{34} + 96^{38}$ is divided by 11.
First, we reduce the bases modulo 11:

$$129 \equiv 8 \pmod{11}$$

So, $129^{34} \equiv 8^{34} \pmod{11}$.
Next, we reduce 96 modulo 11:

$$96 \div 11 = 8 \text{ remainder } 8 \Rightarrow 96 \equiv 8 \pmod{11}$$

So, $96^{38} \equiv 8^{38} \pmod{11}$.
Now we need to find $8^{34} \mod 11$ and $8^{38} \mod 11$.
Using Fermat's Little Theorem, since 11 is a prime number, we know that $a^{p-1} \equiv 1 \pmod{p}$ for any integer a not divisible by 11.
So for $a = 8$ and $p = 11$, we have:

$$8^{10} \equiv 1 \pmod{11}$$

Now, let's find the powers of 8 modulo 11:

$$8^2 = 64 \equiv 64 - 5 \cdot 11 = 64 - 55 = 9 \pmod{11}$$

So, $8^2 \equiv 9 \pmod{11}$.
Now we can find $8^{34} \mod 11$:

$$8^{34} = 8^{2\cdot 17+0} = (8^2)^{17} \equiv 9^{17} \mod 11$$

Since $9 \equiv -2 \pmod{11}$, we have:

$$9^{17} \equiv (-2)^{17} \equiv -2^{17} \pmod{11}$$

We need to find $2^{17} \mod 11$. Using Fermat's Little Theorem, $2^{10} \equiv 1 \mod 11$, so:

$$2^{17} = 2^{10} \cdot 2^7 \equiv 1 \cdot 2^7 \equiv 2^7 \mod 11$$

Now, calculate $2^7 \mod 11$:

$$2^7 = 128 \equiv 128 - 11 \cdot 11 = 128 - 121 = 7 \pmod{11}$$

So, $2^7 \equiv 7 \pmod{11}$, thus:

$$9^{17} \equiv -7 \equiv -7 + 11 = 4 \pmod{11}$$

So, $8^{34} \equiv 4 \pmod{11}$.
Now, let's find $8^{38} \mod 11$:

$$8^{38} = 8^{2\cdot 19+0} = (8^2)^{19} \equiv 9^{19} \mod 11$$

Since $9 \equiv -2 \pmod{11}$, we have:

$$9^{19} \equiv (-2)^{19} \equiv -2^{19} \pmod{11}$$

Now, we need to find $2^{19} \mod 11$:

$$2^{10} \equiv 1 \pmod{11}$$

So,

$$2^{19} = 2^{10} \cdot 2^9 \equiv 1 \cdot 2^9 \equiv 2^9 \mod 11$$

Now, calculate $2^9 \mod 11$:

$$2^9 = 512 \equiv 512 - 46 \cdot 11 = 512 - 506 = 6 \pmod{11}$$

So, $2^9 \equiv 6 \pmod{11}$, thus:

$$9^{19} \equiv -6 \equiv -6 + 11 = 5 \pmod{11}$$

So, $8^{38} \equiv 5 \pmod{11}$.
Now we add the two results:

$$8^{34} + 8^{38} \equiv 4 + 5 \equiv 9 \pmod{11}$$

Thus, the remainder when $129^{34} + 96^{38}$ is divided by 11 is $\boxed{9}$.

MATH EXAMPLE 4

System: Please reason step by step, and put your final answer within $\boxed{}$.

User: Alice and Bob are playing a game. Alice starts first. On Alice's turn, she flips a coin. If she gets a heads, she wins. If not, it becomes Bob's turn. On Bob's turn, he flips a coin. If he gets a tails, he wins. If not, it becomes Alice's turn. What is the probability that Alice wins the game?

Assistant: To determine the probability that Alice wins the game, we need to consider the sequence of turns and the probabilities associated with each player's turn.
First, let's define the probabilities:
The probability that Alice wins on her first turn is the probability that she gets a heads on her coin flip, which is $\frac{1}{2}$.
The probability that Bob wins on his first turn is the probability that he gets tails on his coin flip, which is $\frac{1}{2}$. If Bob gets tails, the game essentially resets with Alice's turn again.
Now, let's denote the probability that Alice wins the game as P.
If Alice doesn't win on her first turn (which happens with probability $\frac{1}{2}$), then it is Bob's turn. The probability that Bob doesn't win on his turn (which happens with probability $\frac{1}{2}$) is $\frac{1}{2}$. If Bob doesn't win, the game returns to the starting point, and the probability that Alice wins from this point onward is still P.
Thus, we can set up the following equation for the probability that Alice wins the game:

$$P = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot \left(\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot P\right)$$

Simplifying the equation:

$$P = \frac{1}{2} \cdot \left(\frac{1}{2} P\right)$$

$$P = \frac{1}{4} P$$

To isolate P, we subtract $\frac{1}{4}P$ from both sides:

$$P - \frac{1}{4}P = 0$$

$$\frac{3}{4}P = 0$$

$$P = \frac{1}{3}$$

Therefore, the probability that Alice wins the game is $\boxed{\frac{1}{3}}$.

User: Check the math solution step-by-step. If you find a mistake: state the wrong step, explain why it's wrong, and end your response with 'The answer is wrong'. If all steps are correct, end your response with 'The answer is correct'.

Assistant: Let's break down the probability of Alice winning step by step.
The probability that Alice wins on her first turn is $\frac{1}{2}$.
If Alice does not win on her first turn, the game essentially resets to the same initial situation. The probability that Alice does not win on her first turn is $\frac{1}{2}$. If she doesn't win on her first turn, it becomes Bob's turn. The probability that Bob doesn't win on his turn is $\frac{1}{2}$, and if Bob doesn't win, the game returns to Alice's turn. The probability that Alice then wins is the same as the probability that she wins the game starting from the beginning, which we'll denote as P.
The probability that Alice wins can be expressed as the sum of probabilities of all scenarios that lead to Alice winning. The probability that Alice wins on her first turn is $\frac{1}{2}$.
The probability that Alice doesn't win on her first turn but then wins on her second turn is the probability that Alice gets tails on her first turn (which is $\frac{1}{2}$), then Bob gets tails (which is $\frac{1}{2}$), and then Alice wins. The probability of this sequence is $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$.
The probability that Alice wins on her third turn is more complex and involves the probability that the sequence of turns leading to Alice winning after two resets. The probability of Alice winning on the third turn is $\left(\frac{1}{2}\right)^3 \times \frac{1}{2} = \frac{1}{8}$.
We can generalize this as an infinite geometric series where the probability that Alice wins on her first turn is $\frac{1}{2}$, and the probability of her winning on subsequent turns is $\left(\frac{1}{2}\right)^n \times \frac{1}{2}$ for each n.
The total probability P that Alice wins is the sum of an infinite geometric series:

$$P = \frac{1}{2} + \left(\frac{1}{2} \times \frac{1}{2}\right) + \left(\frac{1}{2} \times \left(\frac{1}{2}\right)^2\right) + \cdots$$

This series can be summed as:

$$P = \frac{\frac{1}{2}}{1 - \frac{1}{4}} = \frac{\frac{1}{2}}{\frac{3}{4}} = \frac{1}{2} \times \frac{4}{3} = \frac{2}{3}$$

However, the simpler way to look at it is by breaking it into the two mutually exclusive scenarios: 1. Alice wins on her first turn: $\frac{1}{2}$ 2. If Alice doesn't win on her first turn, the game resets to the same initial condition with the same probability distribution. The probability that Alice wins from this new starting point is still P.
Thus, we can set up the equation as:

$$P = \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} \times P$$

$$P = \frac{1}{2} + \frac{1}{4}P$$

Rearranging to solve for P:

$$P - \frac{1}{4}P = \frac{1}{2}$$

$$\frac{3}{4}P = \frac{1}{2}$$

$$P = \frac{1}{2} \times \frac{4}{3} = \frac{2}{3}$$

So, the probability that Alice wins the game is $\frac{2}{3}$.
The answer is wrong.

User: You indicated that your previous answer was wrong. Please provide the correct solution to the math problem.

Assistant: Let's break down the problem step by step to find the probability that Alice wins the game.
Alice wins on her first turn if she gets heads on her coin flip. The probability of Alice winning on her first turn is $\frac{1}{2}$.
If Alice does not win on her first turn, then it's Bob's turn. Bob wins on his first turn if he gets tails on his coin flip. The probability of Bob winning on his first turn is $\frac{1}{2}$.
If Bob doesn't win on his first turn (which happens with probability $\frac{1}{2}$), then it returns to Alice's turn. The game essentially resets with the same probability distribution. Therefore, the probability that Alice wins from this point onward is the same as the original probability P that Alice wins the game.
We can set up the equation to represent the probability that Alice wins the game:

$$P = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2}P$$

Here, $\frac{1}{2}$ is the probability that Alice wins on her first turn, and $\frac{1}{2} \cdot \frac{1}{2}P$ is the probability that the game resets and Alice still has the same probability P to win from this point.

Simplifying the equation:

$$P = \frac{1}{2} + \frac{1}{4}P$$

To isolate P, we subtract $\frac{1}{4}P$ from both sides:

$$P - \frac{1}{4}P = \frac{1}{2}$$

$$\frac{3}{4}P = \frac{1}{2}$$

Now, solve for P by dividing both sides by $\frac{3}{4}$:

$$P = \frac{1}{2} \cdot \frac{4}{3} = \frac{2}{3}$$

Therefore, the probability that Alice wins the game is $\boxed{\frac{2}{3}}$.