REDUCING COGNITIVE OVERHEAD IN TOOL USE VIA MULTI-SMALL-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent progress in multi-agent systems highlights the promise of specialized agents that collaborate through a division of labor. In contrast, most tool-augmented reasoning systems still adopt a single-agent paradigm, where one large model must interleave high-level reasoning with fine-grained tool operations—a process that often leads to cognitive-load interference and unstable outputs. We propose `MSARL` (Multi-Small-Agent Reinforcement Learning), a novel framework that explicitly decouples reasoning from tool execution and interpretation. In `MSARL`, a dedicated reasoning agent focuses on strategic problem decomposition and planning, while a specialized tool agent processes long and complex tool outputs, acting as an adaptive condenser to bridge information gaps. This role-specific separation not only reduces cognitive interference but also accelerates the information flow. To enable effective collaboration, we introduce a hierarchical reinforcement learning approach that uses role-specific and collaboration-based rewards, providing granular feedback to the tool agent and a holistic, trajectory-level signal to the reasoning agent. On mathematical problem-solving with code execution, `MSARL` achieves more stable reasoning and higher final-answer accuracy than strong single-agent baselines. Our findings indicate that this dual-agent architecture significantly mitigates hallucinations and boosts tool invocation tendencies, thereby improving overall robustness. Our method provides a scalable blueprint for building specialized multi-agent system that can tackle complex reasoning tasks. The code for our method is available at: https://anonymous.4open.science/r/msarl-D50D/.

1 INTRODUCTION

The emerging trend in agent-based AI systems is the specialization and collaboration of smaller, role-focused agents(Yang et al., 2023).In tool-integrated reasoning, such division of labor promises gains in efficiency, interpretability, and scalability. Nevertheless, most existing systems still employ a single-agent paradigm, in which one large model sequentially performs high-level reasoning, generates executable tool calls (e.g., code), and interprets results(Xie et al., 2023; Gou et al., 2023; Jin et al., 2025; Qian et al., 2025; Yao et al., 2023). While this integrated design simplifies coordination, it also introduces cognitive load interference: the same model must juggle long-horizon reasoning with precise, low-level tool operations.

We empirically examine this limitation by comparing a single, integrated agent to a decomposed one using identical model architectures and testing data. Despite having access to computational tools, the integrated agent produces fewer correct reasoning paths. This finding suggests that coupling high-level reasoning with tool execution in a single model can degrade the quality of intermediate logical steps.

Motivated by these observations, we present `MSARL` (Multi-Small-Agent Reinforcement Learning), a framework that decouples reasoning from tool use via explicit cognitive-role separation. In `MSARL`, a dedicated reasoning agent decomposes problems into stepwise plans and decides when to invoke tools, while other tool agents each specialize in a specific tool (e.g., code execution, retrieval API, calculator) to process lengthy and complex tool invocation information and results, compress the processed information, and pass it to the reasoning agent, thereby reducing the contextual complexity for the reasoning agent.

To enhance the collaboration capability among agents in handling specialized tasks, the agents are trained jointly through multi-agent reinforcement learning. In our multi-agent reinforcement learning framework, we introduce an innovative collaboration-oriented reward mechanism. Specifically, the reward received by one agent is determined by the quality of the output produced by another agent after processing the information it received from the former. The better the subsequent output of the collaborating agent, the higher the reward for the initiating agent at that step. This design grants agents substantial freedom to explore and discover optimal collaboration patterns for specialized tasks.

We first demonstrate `MSARL` on mathematical problem (MAA, 2025; Lei et al., 2024; MAA, 2023; Hendrycks et al., 2021; He et al., 2024),solving via code execution, where it achieves higher reasoning stability and final-answer accuracy than single-agent baselines(Qwen Team, 2024; Zeng et al., 2025). Beyond mathematics, the architecture naturally generalizes to multi-tool scenarios, offering a scalable blueprint for specialized-agent AI capable of tackling complex reasoning and decision-making tasks.

Our contributions can be summarized as follows:

- We conduct an in-depth empirical analysis of the limitations inherent in single-agent, tool-integrated reasoning systems, showing that coupling high-level reasoning and low-level tool execution within one model can degrade intermediate reasoning quality.
- We propose `MSARL` (Multi-Small-Agent Reinforcement Learning), a novel framework that decouples cognitive roles via a dedicated Reasoning Agent and multiple specialized Tool Agents, equipped with a collaboration-oriented reward mechanism to optimize cooperation and information flow.
- We validate the efficacy of `MSARL` through extensive experiments on mathematical problem solving and multi-tool reasoning tasks, demonstrating superior reasoning stability, final-answer accuracy, and scalability compared to single-agent baselines.

2 Related Work

Tool-Integrated Reasoning. Tool-integrated reasoning (TIR) has emerged as a promising approach to enhance the capabilities of large language models (LLMs). By integrating external tools such as code interpreter (Wang et al., 2023; Gou et al., 2023), search engine (Jin et al., 2025) or LLM-based agents (Wu et al., 2025), TIR serves as an extension to a single executor, allowing models to perform more complex tasks. Despite its great potential, existing TIR approaches exhibit critical limitations. Previous studies distill trajectories from stronger models and perform Supervised Fine-Tuning (SFT), limiting their ability to explore and adapt to optimal reasoning strategies. More recent research show the effectiveness of large-scale reinforcement learning (RL) for TIR with merely outcome rewards (Jaech et al., 2024; DeepSeek-AI, 2025; Qwen Team, 2025). Building on these advances, we focus on mathematical reasoning, a canonical domain for evaluating complex reasoning tasks, and generalize the notion of tool beyond traditional code interpreters to include any auxiliary agent that can support the reasoning process.

Multi-Agent System. Leveraging multi-agent system (MAS) collaboration to complete complex tasks that are difficult to solve by single inference becomes increasingly popular (Han et al., 2024; Tran et al., 2025). In mathematical reasoning, for example, Yuan & Xie (2025) propose an actor-critic architecture in which the critic generates multiple candidate answers and feedback to enable better self-reflection by the actor; Zhang & Xiong (2025) introduce a debating paradigm with diverse agent roles to facilitate fine-grained reasoning through structured disagreement and adjudication. While predefined role interactions are widely adopted in MAS (Lei et al., 2024; Wang et al., 2024; Motwani et al., 2025), some efforts have explored dynamic agent typology and interaction patterns, enabling more flexible and adaptive collaboration (Zhuge et al., 2024; Zhang et al., 2025a; Zhou et al., 2025). However, compared to the success on the single LLM, existing MAS frameworks often lack reliable, fine-grained reward signals for MAS collaboration, relying instead on outputs or self-generated reward mechanisms. Falling into the paradigm of predefined agent collaboration, our work aims to enable the agents to learn how to interact with each other *meanwhile* fine-tune their weights through feedback from cooperation.

Supervision Signals in RL. Existing verification methods in RL can be categorized into three paradigms: outcome-based, process-based, and hybrid supervision. Outcome-based supervision evaluates final outcomes, as seen in the Outcome Reward Model (ORM) (Ouyang et al., 2022). While simple, ORM lacks granularity for intermediate steps, resulting in reward sparsity. Process-based supervision addresses this by providing step-by-step or milestone feedback. Process-supervision Reward Models (PRMs) (Lightman et al., 2023b; Qian et al., 2025) evaluates each reasoning step, enabling precise error correction. However, PRMs rely on costly human-annotated data, which might limit their scalability. Hybrid supervision integrates PRM's step-level quality control with auxiliary rewards (e.g., Instruction Reward Model (IRM) (Luo et al., 2024), which supervises quality of generated instruction) that supervise data quality from orthogonal angles (e.g., instruction quality). Although this complementary approach mitigates single-source limitations, it introduces optimization conflicts requiring careful calibration. These reward signals motivate diverse policy optimization methods such as PPO (Schulman et al., 2017), DPO (Rafailov et al., 2023), GRPO (Shao et al., 2024). Among them, GRPO's flexible reward function enables adaptation to diverse objectives, such as assigning weights to subtasks (Yu et al., 2024) or constraining tool use frequency (Li et al., 2025). In this work, we extend GRPO to enhance multi-agent collaboration in mathematical problem solving through a simple yet effective reward design.

3 METHODOLOGY

In this section, we start from a pilot investigation to see whether augmenting a single agent with code execution improves mathematical reasoning, or instead may introduce additional cognitive burden. Then we present the detailed design for training models by our proposal, i.e., `MSARL`, covering (1) rollout framework; (2) reward design; (3) training strategy.

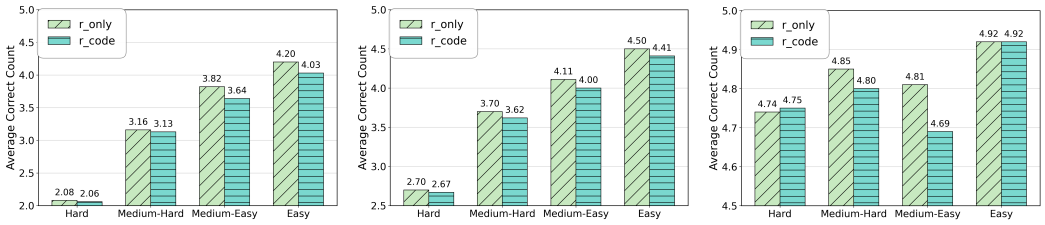
3.1 COGNITIVE OVERHEAD IN SINGLE-AGENT REASONING

Setup. We conduct experiments on the non-reasoning model Qwen2.5-3B-Instruct (Qwen Team, 2024), the reasoning model Qwen3-4B (Qwen Team, 2025) and a mathmatical models Qwen2.5-Math-1.5B-Instruct (Yang et al., 2024) for mathmatical reasoning tasks. Specifically, two prompting regimes are considered with the same backbone model: (i) *reasoning-only* (`r_only`), in which the model is instructed to solve problems exclusively through step-by-step natural language reasoning; (ii) *reasoning with code* (`r_code`), in which the model is encouraged to interleave Python code blocks with natural language reasoning, thereby allowing intermediate computations to be executed. Experiments are conducted on the MATH-500 test split (Lightman et al., 2023a), a benchmark whose problems are naturally amenable to solution verification through executable code. For each problem, we draw $N = 5$ independent samples per regime using nucleus sampling ($p = 0.95$, temperature $= 0.7$). To assess solution validity, we employ DeepSeek-R1 (DeepSeek-AI, 2025) as the judge to evaluate whether a reasoning trajectory, disregarding minor arithmetic slips, constitutes a logically coherent path that would yield the correct answer under flawless execution. The complete prompts for two prompting strategies and model judge can be found in Appendix A.

Results. To analyze performance across different reasoning difficulty levels, we categorize MATH-500 problems into four groups based on their inherent difficulty levels (ranging from 1 to 5) and their respective proportions in the dataset[1]: `Hard` (Level 5, 26.8%), `Medium-Hard` (Level 4, 25.6%), `Medium-Easy` (Level 3, 21%), and `Easy` (Levels 1–2, 26.6%). As shown in Figure 1, the `r_code` regime generally underperforms the `r_only` regime, exhibiting accuracy gaps of 0.02 to 0.18 across all difficulty levels. The exception is Qwen3-4B, which demonstrates superior average performance on mathematical tasks, as expected given its strong reasoning capabilities in thinking mode (Qwen Team, 2025). The degradation is most pronounced on problems of medium-difficulty (`Medium-Hard`: -0.08, `Medium-Easy`: -0.18), while the performance gap is smaller on the more complex `Hard` problems. This pattern suggests that the additional cognitive load induced by generating, executing, and integrating code fragments is particularly disruptive when problems require a moderate level of reasoning complexity, where code execution might interfere with the initial problem formulation. To further illustrate this phenomenon, we provide qualitative analysis of a medium-difficulty example in Appendix B, contrasting model outputs under the two prompt-

[1] https://huggingface.co/datasets/HuggingFaceH4/MATH-500

ing regimes. These findings indicate that a single agent can struggle to effectively balance internal reasoning with external tool-use, suggesting a fundamental limitation in its ability to manage the cognitive load of both tasks simultaneously.



(a) Qwen2.5-3B-Instruct (b) Qwen2.5-Math-1.5B-Instruct (c) Qwen3-4B

Figure 1: Model performance on Math-500 under two prompting regimes.

3.2 Dual-Agent Framework for Decoupled Reasoning and Tool Use

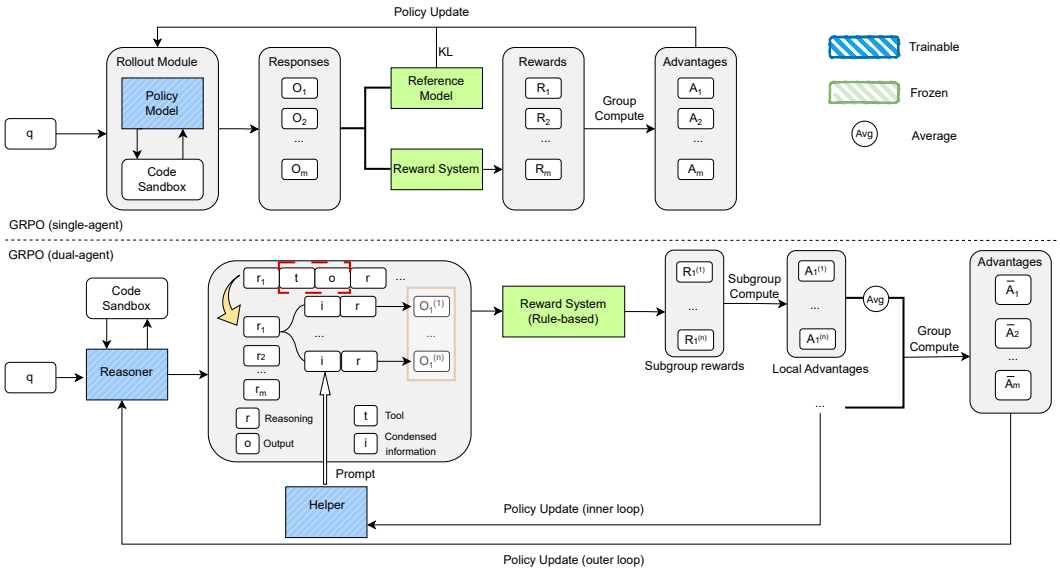


Figure 2: Overview of our method

Preliminaries and Notations. Motivated by the observations in Section 3.1, we propose MSARL, a *dual-agent framework* designed to mitigate cognitive interference by explicitly decoupling the responsibilities of reasoning and tool-use. Figure 2 illustrates the framework of MSARL. In the proposed dual-agent system, one serves as a reasoning agent *Reasoner* and the other as a tool agent *Helper*. In the context of LLM policy optimization, let π_{reason} be the reasoning LLM, π_{tool} be the tool-assisting LLM. For each question q in a given set Q, the *Reasoner* generates one or more independent reasoning trajectories. Each reasoning trajectory consists of a sequence of steps $< r, t, o >$. At step k, the *Reasoner* may produce either a natural language reasoning segment r_k or a tool call t_k. When a tool call is executed, it returns the tool output o_k. A key step in our framework is the interaction with the *Helper*. The pair $< t_k, o_k >$ is processed by the *Helper* model π_{tool} to generate a condensed, structured interpretation, denoted as i_k. This interpretation i_k is then passed back to the *Reasoner*, which uses the full context, including r_k and i_k, to generate the subsequent reasoning step r_{k+1}. This iterative process continues until the *Reasoner* produces a final answer, denoted as O.

Rollout Framework. To enable the model to autonomously generate reasoning traces and tool calls, we utilize the prompts as shown in Figure 3. Following Li et al. (2025), when a code termination identifier (```output) is detected, the *Reasoner* pauses generation; the latest code block

is extracted for code execution in the code sandbox; both the code `Code` and the structured execution result `Output` are wrapped by training prompt for tool agent, resulting in *Helper*'s output inserted into the `<answer>` field. The *Reasoner*'s final answer is indicated within a box (e.g., $\boxed{final\_answer}$). We observed the agent-to-agent communication during the rollout process introduces significant GPU idle time. To maximize the training efficiency, we introduced a hyperparameter, C, which is the maximum number of tool calls the model can make during a single response generation. Once this threshold is exceeded, the system ignores further code execution requests, forcing the model to switch to pure-text reasoning mode. To better illustrate the training dynamics, we provide a full rollout example in Appendix C.

Training Prompt for Reasoning Agent

Please integrate natural language reasoning with programs to solve the problem above, and put your final answer within boxed {}. Problem: `Question`

Training Prompt for Tool Agent

Describe in one sentence what this code does, and in one sentence explain the meaning of its output.
Wrap your answer in `<answer></answer>`
Python code: `Code`
Output: `Output`

Figure 3: Training prompt templates for `MSARL`

Reward Design. A critical challenge in optimizing multi-step reasoning agents is the sparsity of terminal reward signals, which often fail to provide sufficient guidance for intermediate reasoning steps. To address this issue, our reward design strategically applies the outcome-based reward at the most decisive juncture, that is, the tool-use interpretation stage as identified in Section 3.1. Specifically, the binary success reward (Eq. 1) is granted based on the output of the *Helper* agent (π_{tool}). It checks whether the *Reasoner* model output is both calculated and formatted correctly, as specified by the ground truth. This approach reframes the final outcome evaluation as a *dense and immediate milestone reward* for effective tool utilization. Crucially, during joint optimization, this reward signal propagates back to the *Reasoner* policy (π_{reason}). This process operates as an implicit regularization mechanism, incentivizing the *Reasoner* to generate logically sound reasoning traces that culminate in correct and executable tool invocations.

$$\mathcal{R} = \begin{cases} 1, & \text{if the answer is correct and boxed} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

***Reasoner-Helper* Framework.** Our methodology is built upon a hierarchical reinforcement learning framework designed to jointly optimize both the reasoning policy (π_{reason}) and the tool policy (π_{tool}). To tune the model with structured rewards, we employ GRPO (Shao et al., 2024) that introduces advantage normalization within grouped samples. For brevity, assume that π_{reason} parallelly generates m reasoning trajectories up to the point of its final tool invocation from the question q. For such a specific clipped reasoning trajectory τ_k containing tool execution results. At this juncture, the *Helper* policy π_{tool} generates a set of n diverse interpretations of the tool's output, denoted as $\{i_k^{(1)}, i_k^{(2)}, \ldots, i_k^{(n)}\}$. Each of these interpretations is then passed back to the *Reasoner*. The *Reasoner* acts as a proxy to complete the reasoning process, generating a final conclusive segment for each interpretation. This branching process results in a set of n distinct final answers, $\{O_k^{(1)}, O_k^{(2)}, \ldots, O_k^{(n)}\}$. Next, the reward system in Eq. 1 scores each output $O_k^{(j)}$ based on its correctness relative to the ground truth (GT), yielding a reward value $R_k^{(j)}$. Although the reward is computed on the final answer $O_k^{(j)}$, it acts as a direct performance signal for the corresponding interpretation $i_k^{(j)}$ that produced it. These outputs and their corresponding reward values form an

evaluation subgroup G_k:

$$G_k = \{(O_k^{(1)}, R_k^{(1)}), (O_k^{(2)}, R_k^{(2)}), \ldots, (O_k^{(n)}, R_k^{(n)})\}$$

We then calculate the normalized advantage $A_k^{(j)}$ for each output relative to the mean μ_k and standard deviation σ_k of rewards within the subgroup G_k. A small constant η is added for numerical stability. The normalized advantage for each output is then defined as:

$$A_k^{(j)} = \frac{R_k^{(j)} - \mu_k}{\sigma_k + \eta} \tag{2}$$

$$\mu_k = \frac{1}{n}\sum_{j=1}^{n} R_k^{(j)}, \quad \sigma_k = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(R_k^{(j)} - \mu_k)^2} \tag{3}$$

Since the ultimate goal is to optimize the generation of the entire reasoning trajectory, we perform average pooling on the advantage values computed in the clipped reasoning trajectory τ_k to obtain an aggregated advantage value $\bar{A}_k$ that represents the overall quality of the entire trajectory:

$$\bar{A}_k = \frac{1}{n}\sum_{j=1}^{n} A_k^{(j)} \tag{4}$$

The training objectives, adapted from GRPO, are designed to separately update the parameters of the tool policy (θ_{tool}) and the reasoning policy (θ_{reason}). First, we optimize the tool policy π_{tool}. The goal is to encourage the *Helper* agent to generate interpretations that lead to correct final answers. The normalized advantage $A_k^{(j)}$ is applied at the token level to the corresponding interpretation $i_k^{(j)}$ that leads to the final outcome $O_k^{(j)}$. The learning objective of π_{tool} is formulated as:

$$\mathcal{J}(\theta_{\text{tool}}) = \mathbb{E}\tau_k, i_k^{(j)} \left[\frac{1}{n}\sum_{j=1}^{n} \frac{1}{|i_k^{(j)}|} \sum_{t=1}^{|i_k^{(j)}|} \min\left(r_t^{(j)}(\theta_{\text{tool}}) A_k^{(j)}, \text{clip}(r_t^{(j)}(\theta_{\text{tool}}), 1-\epsilon, 1+\epsilon) A_k^{(j)} \right) \right] \tag{5}$$

where $r_t^{(j)}(\theta_{\text{tool}}) = \frac{\pi_{\text{tool},\theta}(i_{k,t}^{(j)} | \tau_k, i_{k,<t}^{(i)})}{\pi_{\text{tool,old}}(i_{k,t}^{(j)} | \tau_k, i_{k,<t}^{(j)})}$ is the probability ratio for the t-th token of the j-th interpretation, and the expectation is over the distribution of trajectories and interpretations.

Next, we optimize the reasoning policy π_{reason}. The objective is to guide the *Reasoner* to generate trajectories that are more likely to result in high-reward outcomes after tool interpretation. The aggregated advantage $\bar{A}_k$, representing the average quality of a reasoning path τ_k, is applied to every token within that path. The objective is thus defined as:

$$\mathcal{J}(\theta_{\text{reason}}) = \mathbb{E}\tau_k \left[\frac{1}{m}\sum_{k=1}^{m} \frac{1}{|\tau_k|} \sum_{t=1}^{|\tau_k|} \min\left(r_t^{(k)}(\theta_{\text{reason}}) \bar{A}_k, \text{clip}(r_t^{(k)}(\theta_{\text{reason}}), 1-\epsilon, 1+\epsilon) \bar{A}_k \right) \right] \tag{6}$$

where $rt^{(k)}(\theta_{\text{reason}}) = \frac{\pi_{\text{reason},\theta}(\tau_{k,t} | q, \tau_{k,<t})}{\pi_{\text{reason,old}}(\tau_{k,t} | q, \tau_{k,<t})}$ is the probability ratio for the t-th token of the k-th reasoning trajectory.

Recent work (Qian et al., 2025; Zhang et al., 2025b) suggest that the omission of KL penalty term against a reference model can encourage the model to more freely adapt its behavior to our custom response format and structured reward signals. Following their implementation, we remove the term to simplify the training pipeline while gain comparable performance in practice. We summarize the process for advantage estimation and policy update in Appendix D.

4 EXPERIMENTS

4.1 IMPLEMENTATION DETAILS

Datasets and Metrics. The training utilizes approximately 1.2w queries from all difficulty levels of the MATH training dataset (Hendrycks et al., 2021). We evaluate the proposed method on widely used mathmatical benchmarks, including AIME 2024 (Li et al., 2024), AIME25 (MAA, 2025)[2], MATH-500 (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), AMC23 (MAA, 2023). We report Pass@1 as in (Cui et al., 2025). In addition, to measure stability and consistency of model generated responses, we include Pass@8 and Maj@8 on all benchmarks, following prior work (Yang et al., 2024; Zhang et al., 2025c).

Models and Baselines. We choose Qwen2.5-Math-Instruct (Yang et al., 2024) series models as backbones for their superior mathmatical capabilities. Specifically, we use Qwen2.5-Math-1.5B-Instruct as the backbone of the reasoning agent, and Qwen2.5-1.5B-Instruct as the backbone of the tool agent. We compare our method mainly against reinforcement learning-based approaches, most of which are single-agent architecture.

Training and Evaluation Details. We use Sandbox Fusion as the code interpreter during training and evaluation. Due to the dual-agent framework, the global batch size is set to 1 by default. The *Reasoner* generates 3 samples per question and the *Helper* uses a rollout size of 3. To maximize training efficiency, the default maximum number of tool calls C is set to 1. All models are RL-tuned with a cold start. For evaluation, we use greedy decoding (temperature = 0) across all models. For a fair comparison, we set the maximum number of tool calls to 1 during inference.

4.2 MAIN RESULTS

▷ MSARL-1.5B achieves the highest average pass@1 accuracy of 55.9% across all five datasets, as shown in Table 1. This represents a substantial improvement of 5.9% over the strongest baseline, Qwen2.5-Math-1.5B-Instruct-TIR (50.0%). MSARL-1.5B also surpasses other methods on challenging MATH500 (77.6%) and Olympiad (49.0%) datasets. This clear advantage confirms the effectiveness of introducing agents joint learning into a multi-agent system. Furthermore, our method show remarkable parameter efficiency. Our 1.5B models surpasses most competing 7B models by a considerable margin. For instance, it outperforms SimpleRL-Zero and Qwen2.5-Math-7B-Instruct by $14.2, 16.1$ in pass@1 accuracy. This result implies that our MSARL approach provides a more effective and efficient path to enhancing mathematical reasoning in language models than simply scaling up model size or applying standard fine-tuning techniques.

Table 1: Comparison of different models testing accuracy on mathematical benchmarks with Pass@1. The best two performance are **bold** and underlined.

Model	AIME24	AIME25	MATH500	Olympiad	AMC23	Avg
Models based on Qwen2.5-Math-1.5B-Base						
Qwen2.5-Math-1.5B-Instruct	10.0	10.0	66.0	31.0	62.5	35.9
Qwen2.5-Math-1.5B-Instruct-TIR	<u>23.3</u>	**20.0**	<u>75.6</u>	<u>48.5</u>	<u>62.5</u>	<u>50.0</u>
Models based on Qwen2.5-Math-7B-Base						
Qwen2.5-Math-7B-Instruct	10.0	16.7	74.8	32.4	**65.0**	39.8
Qwen2.5-Math-7B-Instruct-TIR	20.0	6.7	70.4	45.0	50.0	34.2
SimpleRL-Zero	**30.0**	<u>20.0</u>	66.8	29.0	62.5	41.7
Eurus-2-7B-PRIME	10.0	13.3	62.8	42.1	50.0	35.6
MSARL-1.5B (Untrained)	23.3	20.0	74.4	47.0	62.5	<u>52.6</u>
MSARL-1.5B (Ours)	16.7	16.7	**77.6**	**49.0**	57.5	**55.9**$_{+5.9}$

▷Our method generally shows strong performance, securing either the best or second-best results in a majority of the pass@8 and maj@ metrics as summarized in Table 2. Specifically, it achieves the best maj@8 score on MATH500 and the top scores for both pass@8 and maj@8 on the Olympiad dataset. It also ties for the best pass@8 on AMC23 and has the highest maj@8 score on that same

[2]https://modelscope.cn/datasets/TIGER-Lab/AIME25

dataset. As the maj@8 metric is a more robust indicator of a model's consistency and confidence, MSARL strong performance on maj@8 across most datasets suggests that its dual-agent approach leads to more reliable and stable outputs.

Table 2: Comparison of different models on mathematical benchmarks with Pass@8 and Maj@8. The best two performance are **bold** and underlined.

Model	AIME24		AIME25		MATH500		Olympiad		AMC23	
	pass@8	maj@8	pass@8	maj@8	pass@8	maj@8	pass@8	maj@8	pass@8	maj@8
Qwen2.5-Math-1.5B-Instruct-TIR	40.0	20.0	**40.0**	**30.0**	**93.4**	81.6	66.5	54.3	87.5	60
Qwen2.5-Math-7B-Instruct-TIR	46.7	26.7	26.7	13.3	88.8	78.4	63.9	48.1	80	67.5
SimpleRL-Zero	**50.0**	**30.0**	26.7	20.0	90.2	82.0	-	-	85.0	67.5
Eurus-2-7B-PRIME	46.7	20.0	36.7	16.7	90.2	73.4	60.4	40.2	85.0	57.5
MSARL-1.5B (Ours)	40.0	23.3	36.7	20.0	92	**82.6**	**77.3**	**54.7**	**87.5**	**72.5**

4.3 ABLATION STUDY AND EXTRA INVESTIGATION

Contribution of Joint Training. To verify that the superior performance of our method originates from the proposed joint optimization strategy, rather than merely from the inherent structure of a dual-agent workflow, we replace our two trained agents with their respective untrained models and evaluate this configuration. A performance degradation -3.6 in the untrained setup is observed in the penultimate row in Table 1, which confirms the effectiveness of our joint optimization framework. Interestingly, we observe that even the dual-agent architecture with untrained models surpasses the strongest single LLM baseline in a zero-shot setting. This validates the effectiveness of the decoupling strategy on complex reasoning problems in its own right.

Evaluation across Different Training Steps. To analyze the training dynamics and convergence of our framework, we evaluated its performance at various training checkpoints. Figure 4 plots the average Pass@1 accuracy on our validation benchmarks as a function of the number of training steps. Figure 4 reveals a rapid performance improvement during the early stages of training, which is followed by a steady but diminishing rate of gain as training progresses. The frameworks's performance begins to saturate around the 2k training step mark, indicating stable convergence without signs of overfitting on the test data. This analysis shows the robustness of our training process and justifies our selection of the final model checkpoint for the main evaluation.

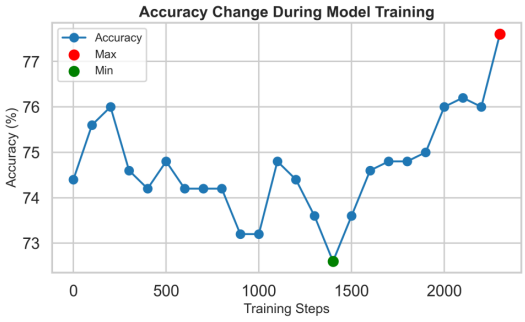


Figure 4: The model's performance (Average Pass@1) at different training checkpoints. Performance saturates after 2k steps.

Reward Dynamics During Training. To provide insight into the learning process of our framework, we analyze the reward dynamics during training. Figure 5 plots the average reward score obtained by the agents as a function of training steps. The curve, smoothed with a moving average for clarity, first shows a stable increase indicating that our agent consistently improves its policy to maximize rewards. The absence of significant volatility or collapses in the reward curve confirms the robustness of our training configuration and hyperparameter settings. The reward begins to plateau in the later stages, suggesting that the agent is converging to an effective and stable policy.

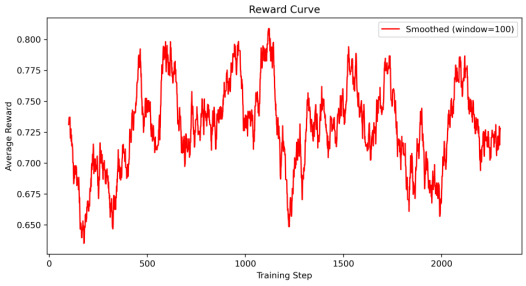


Figure 5: Average reward score during training. The consistent upward trend demonstrates successful and stable learning, with the policy converging in the later stages.

5 CONCLUSION

In this paper, we first identify a key challenge: the cognitive load interference inherent in the single-agent paradigm for tool-integrated reasoning. We then propose MSARL, a novel dual-agent framework that explicitly decouples high-level reasoning from low-level tool interpretation. This division of labor, powered by two specialized agents, significantly enhances the efficiency of information flow and addresses the identified interference. Specifically, MSARL leverages a reasoning agent for macro-level problem planning and a tool agent for micro-level, adaptive tool output interpretation. To optimize these agents jointly, we introduce a hierarchical reinforcement learning approach based on GRPO. This method uses normalized advantages to provide a granular, high-fidelity training signal to the tool agent and aggregated advantages to provide a consistent, trajectory-level signal to the reasoning agent. Extensive experiments on mathematical problem-solving tasks demonstrate that MSARL consistently achieves superior performance and higher reasoning stability compared to single-agent baselines. Furthermore, the modular nature of MSARL naturally generalizes to multi-tool scenarios, offering a scalable blueprint for future agent-based systems. Future work could explore extending MSARL to scenarios with more diverse and complex tool sets and information sources.

REFERENCES

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards, 2025. URL https://arxiv.org/abs/2502.01456.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023. URL https://arxiv.org/abs/2309.17452.

Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. LLM multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*, 2024. URL https://arxiv.org/abs/2402.03578.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Zhen Hu, Shengding andThai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

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. *NeurIPS*, 2021.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Carney, Alex Beutel, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. URL https://arxiv.org/abs/2412.16720.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09516.

Bin Lei, Yi Zhang, Shan Zuo, Ali Payani, and Caiwen Ding. MACM: Utilizing a multi-agent system for condition mining in solving complex mathematical problems. *arXiv preprint arXiv:2404.04735*, 2024. URL https://arxiv.org/abs/2404.04735.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl, 2025. URL https://arxiv.org/abs/2503.23383.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023a.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023b. URL https://arxiv.org/abs/2305.20050.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, and Dongmei Zhang. WizardMath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=mMPMHWOdOy.

MAA. American mathematics competitions, 2023. URL https://www.maa.org/math-competitions. Accessed: 2025-09-17.

MAA. 2025 AIME i problems, 2025. URL https://artofproblemsolving.com/wiki/index.php/2025_AIME_I?srsltid=AfmBOoof5gaaqlt3-l6LH7Tt6qmJZtl_2PQEDYlLFlMqhq9dLL8FMCRR. Accessed: 2025-09-17.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent LLM training. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=jXP9bgFack.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Paul Mishkin, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs, 2025. URL https://arxiv.org/abs/2504.13958.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

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

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

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*, 2025. URL https://arxiv.org/abs/2501.06322.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024. URL https://arxiv.org/abs/2406.04692.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in LLMs for enhanced mathematical reasoning. *arXiv preprint arXiv:2310.03731*, 2023. URL https://arxiv.org/abs/2310.03731.

Junde Wu, Jiayuan Zhu, Yuyuan Liu, Min Xu, and Yueming Jin. Agentic reasoning: A streamlined framework for enhancing LLM reasoning with agentic tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.acl-long.1383.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, et al. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*, 2023.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *arXiv preprint arXiv:2410.07745*, 2024.

Yurun Yuan and Tengyang Xie. Reinforce llm reasoning through multi-agent reflection. In *International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=6k3oFS3Lbl.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-Designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2025a. URL https://arxiv.org/abs/2410.11782.

Shaowei Zhang and Deyi Xiong. Debate4MATH: Multi-agent debate for fine-grained reasoning in math. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 5: Findings of ACL)*, pp. 14524–14537, Vancouver, Canada, July 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.findings-acl.862.

Xiaoying Zhang, Hao Sun, Yipeng Zhang, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback, 2025b. URL https://arxiv.org/abs/2506.03106.

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, 2025c. URL https://arxiv.org/abs/2501.07301.

Hongyi Zhou, Haoran Geng, Xingcheng Xue, Lei Kang, Yuhang Qin, Zihan Wang, et al. Reso: A reward-driven self-organizing llm-based multi-agent system for reasoning tasks. *arXiv preprint arXiv:2503.02390*, 2025. URL https://arxiv.org/abs/2503.02390.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024. URL https://arxiv.org/abs/2402.16823.

A MOTIVATION PROMPTS

Reasoning-Only Prompt. The following prompt is used during all reasoning-only inference experiments:

Reasoning-Only Prompt

System: You are an IMO medalist mathematician.
User:
Question: `<Question Content>`
Instructions:
1. Think step-by-step in natural language to derive a complete solution.
2. Provide ONLY the final mathematical answer in the last line, formatted as:
ANSWER: `<Your Answer>`
3. Do NOT write or reference any code. Stop after giving the answer.

Reasoning with Code Prompt. The following prompt is used during all reasoning with code inference experiments:

Reasoning with Code Prompt

System: You are an IMO medalist mathematician who can also run Python (with sympy, itertools, math, random).
User:
Question: `<Question Content>`
Instructions:
1. Think step-by-step in natural language.
2. Whenever useful, place Python code inside "'python ..."' cells. The code will be executed; you can refer to its output.
3. After you finish reasoning, output your final answer on a separate line: ANSWER: `<Your Answer>`

Prompt for Reasoning Path Critique. We adopt a prompt to enable DeepSeek-R1 to evaluate the correctness of generated responses using the above two prompting regimes:

Prompt for Reasoning Path Critique

System: You are an experienced automatic grading assistant. Only output the result for the question in the specified format.
User:
I have a question: `<Question Content>`
Below are 5 different answers to this question, all from the same small model.
For each answer, please judge whether the answer contains a correct approach (i.e., Is the line of thinking or analysis correct? It doesn't need to be a fully correct final answer, just whether the reasoning process is on the right track).
Count how many out of the 5 responses have a correct approach, and ONLY output the number of correct reasoning answers in the format: #number# (for example, #3# means 3 out of 5 answers are correct in reasoning).
Do not provide any explanations, just output the result in the requested format.
Here are the answers:
`<Answer List>`

B MOTIVATION QUALITATIVE EXAMPLE

This example demonstrates a comparison between two different prompting methods used to solve a medium-hard (level 4) math problem with the Qwen2.5-Math-1.5B-Instruct model:

Question
Given

$$\mathbf{A} = \begin{pmatrix} 3 & 7 \\ -1 & -2 \end{pmatrix},$$

compute $\mathbf{A}^{27} + \mathbf{A}^{31} + \mathbf{A}^{40}$.

Solution (Ground Truth)
Note that

$$\mathbf{A}^2 = \begin{pmatrix} 2 & 7 \\ -1 & -3 \end{pmatrix}$$

and

$$\mathbf{A}^3 = \mathbf{A}\mathbf{A}^2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = -\mathbf{I}.$$

Then

$$\begin{aligned} \mathbf{A}^{27} + \mathbf{A}^{31} + \mathbf{A}^{40} &= (\mathbf{A}^3)^9 + (\mathbf{A}^3)^{10}\mathbf{A} + (\mathbf{A}^3)^{13}\mathbf{A} \\ &= (-\mathbf{I})^9 + (-\mathbf{I})^{10}\mathbf{A} + (-\mathbf{I})^{13}\mathbf{A} \\ &= -\mathbf{I} + \mathbf{A} - \mathbf{A} \\ &= -\mathbf{I} = \boxed{\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}}. \end{aligned}$$

Solution 1: Incorrect Response (Reasoning with Code)

To solve the problem, we need to compute $\mathbf{A}^{27} + \mathbf{A}^{31} + \mathbf{A}^{40}$ where $\mathbf{A} = \begin{pmatrix} 3 & 7 \\ -1 & -2 \end{pmatrix}$.

The steps are as follows:

1. Find the eigenvalues and eigenvectors of $\mathbf{A}$.
2. Diagonalize the matrix $\mathbf{A}$ if possible, i.e., express $\mathbf{A}$ as $\mathbf{PDP}^{-1}$, where $\mathbf{D}$ is a diagonal matrix of eigenvalues and $\mathbf{P}$ is a matrix of corresponding eigenvectors.
3. Compute $\mathbf{A}^{27}$, $\mathbf{A}^{31}$, and $\mathbf{A}^{40}$ using the diagonalized form.
4. Sum the resulting matrices.

Let's implement this in Python using sympy.

```python
import sympy as sp

# Define the matrix A
A = sp.Matrix([[3, 7], [-1, -2]])

# Find the eigenvalues and eigenvectors
# Note: diagonalize() returns (P, D) where A = P * D * P.inv()
P, D = A.diagonalize()

# Compute A^27, A^31, and A^40 using the diagonalized form
A_27 = P * D**27 * P.inv()
A_31 = P * D**31 * P.inv()
A_40 = P * D**40 * P.inv()

# Compute the final result
result = A_27 + A_31 + A_40

# Print the final result
print(result)
```

```output
sympy.matrices.common.MatrixError: Matrix is not diagonalizable
```

The error indicates that the matrix $\mathbf{A}$ is not diagonalizable. This means that we cannot directly diagonalize $\mathbf{A}$ and use the diagonalization method to compute $\mathbf{A}^{27}$, $\mathbf{A}^{31}$, and $\mathbf{A}^{40}$.
Instead, we can use the Cayley-Hamilton theorem, which states that a matrix satisfies its own characteristic equation. The characteristic polynomial of $\mathbf{A}$ is given by:

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \begin{vmatrix} 3-\lambda & 7 \\ -1 & -2-\lambda \end{vmatrix} = (3-\lambda)(-2-\lambda) + 7 = \lambda^2 - \lambda + 2.$$

According to the Cayley-Hamilton theorem, $\mathbf{A}^2 - \mathbf{A} + \mathbf{I} = \mathbf{0}$, which implies:

$$\mathbf{A}^2 = \mathbf{A} - \mathbf{I}.$$

Using this relation, we can express higher powers of $\mathbf{A}$ in terms of $\mathbf{A}$ and $\mathbf{I}$.
Let's use this relation to compute $\mathbf{A}^{27}$, $\mathbf{A}^{31}$, and $\mathbf{A}^{40}$.

1. Compute $\mathbf{A}^{27}$:
$$\mathbf{A}^{27} = \mathbf{A}^{27 \mod 2} = \mathbf{A}^1 = \mathbf{A}.$$
2. Compute $\mathbf{A}^{31}$:
$$\mathbf{A}^{31} = \mathbf{A}^{31 \mod 2} = \mathbf{A}^1 = \mathbf{A}.$$
3. Compute $\mathbf{A}^{40}$:
$$\mathbf{A}^{40} = \mathbf{A}^{40 \mod 2} = \mathbf{A}^0 = \mathbf{I}.$$

Thus, $\mathbf{A}^{27} + \mathbf{A}^{31} + \mathbf{A}^{40} = \mathbf{A} + \mathbf{A}$ ...(truncated)

Solution 2: Correct Response (Reasoning Only)
To solve the problem of computing $\mathbf{A}^{27} + \mathbf{A}^{31} + \mathbf{A}^{40}$, we need to first understand the behavior of the matrix $\mathbf{A}$ under exponentiation. We will use the characteristic polynomial of the matrix $\mathbf{A}$ to find its eigenvalues and then use these eigenvalues to find a pattern in the powers of $\mathbf{A}$.
The characteristic polynomial of $\mathbf{A}$ is given by:

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \begin{vmatrix} 3-\lambda & 7 \\ -1 & -2-\lambda \end{vmatrix} = (3-\lambda)(-2-\lambda) + 7 = \lambda^2 - \lambda + 1.$$

The eigenvalues λ_1 and λ_2 are the roots of the characteristic polynomial $\lambda^2 - \lambda + 1 = 0$. Solving this quadratic equation, we get:

$$\lambda = \frac{1 \pm \sqrt{1-4}}{2} = \frac{1 \pm \sqrt{-3}}{2} = \frac{1 \pm i\sqrt{3}}{2}.$$

Let $\lambda_1 = \frac{1+i\sqrt{3}}{2}$ and $\lambda_2 = \frac{1-i\sqrt{3}}{2}$.
Since $\mathbf{A}$ is a 2x2 matrix with eigenvalues λ_1 and λ_2, we can express $\mathbf{A}$ in the form $\mathbf{A} = \mathbf{PDP}^{-1}$, where $\mathbf{D}$ is the diagonal matrix of eigenvalues and $\mathbf{P}$ is the matrix of eigenvectors.
However, instead of explicitly finding $\mathbf{P}$ and $\mathbf{P}^{-1}$, we can use the fact that $\mathbf{A}^n = \mathbf{PD}^n\mathbf{P}^{-1}$ for any integer n.
Since $\mathbf{D}$ is diagonal, $\mathbf{D}^n$ is simply the diagonal matrix with λ_1^n and λ_2^n on the diagonal.
Observe that $\lambda_1 = e^{i\pi/3}$ and $\lambda_2 = e^{-i\pi/3}$ are primitive 6th roots of unity. Therefore:

$$\lambda_1^3 = e^{i\pi} = -1, \quad \lambda_2^3 = e^{-i\pi} = -1.$$

Since $\mathbf{A}^3$ has eigenvalues $\lambda_1^3 = -1$, $\lambda_2^3 = -1$, and since $\mathbf{A}^3$ is diagonalizable, it must be:

$$\mathbf{A}^3 = -\mathbf{I}.$$

More precisely:

$$\mathbf{A}^n = \begin{cases} (-1)^k\mathbf{I}, & n = 3k, \\ (-1)^k\mathbf{A}, & n = 3k+1, \\ (-1)^k\mathbf{A}^2, & n = 3k+2. \end{cases}$$

(truncated)

Analysis. The initial implementation of Solution 1 follows a mathematically sound strategy: attempt to diagonalize the matrix $\mathbf{A}$ and use the decomposition $\mathbf{A} = \mathbf{PDP}^{-1}$ to compute high powers efficiently via $\mathbf{A}^n = \mathbf{PD}^n\mathbf{P}^{-1}$. The Python code using `sympy` is syntactically correct and logically coherent. This approach would succeed if $\mathbf{A}$ were diagonalizable over the real numbers. However, the code raises the exception: "sympy.matrices.common.MatrixError: Matrix is not diagonalizable" This error disrupts the solution path and triggers a cascade of flawed reasoning. Although the matrix

$$\mathbf{A} = \begin{pmatrix} 3 & 7 \\ -1 & -2 \end{pmatrix}$$

has distinct eigenvalues $\lambda = \frac{1 \pm i\sqrt{3}}{2}$ (roots of $\lambda^2 - \lambda + 1 = 0$), it is diagonalizable *over the complex numbers*. The error arises because `diagonalize()` may fail to find a real eigenvector matrix, not because diagonalization is impossible in principle. Instead of handling this gracefully—such as by manually computing eigenvectors over $\mathbb{C}$ or switching to an alternative method—the solution incorrectly interprets the error as a complete dead end. This leads to a series of hallucinated steps:

- The characteristic polynomial is incorrectly stated as $\lambda^2 - \lambda + 2$, though the subsequent derivation $\mathbf{A}^2 = \mathbf{A} - \mathbf{I}$ implies the correct polynomial $\lambda^2 - \lambda + 1$ was intended.
- Most critically, the solution **fabricates a false periodicity**, claiming:

  $$\mathbf{A}^{27} = \mathbf{A}^{27 \mod 2} = \mathbf{A}, \quad \mathbf{A}^{40} = \mathbf{I}, \quad \text{etc.}$$

  This "mod 2 reduction" has no mathematical basis and is a hallucination induced by the earlier computational failure.

In contrast, Solution 2 presents a sophisticated and mathematically sound approach to computing the problem, even without code invocation and execution. It avoids brute-force computation by leveraging the algebraic structure of the matrix through eigenvalues and periodicity. The solution is truncated (due to the limit on generation max tokens), but the framework is complete.

In summary, while the initial code logic is correct, the Solution 1 lacks **mathematical robustness** in handling the exception. The interruption caused by the diagonalization error derails the reasoning, resulting in hallucinated algebraic rules and a fundamentally incorrect final answer.

C SUCCESSFUL ROLLOUT EXAMPLE

Question
If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2)+f(-1)+f(0)$? Express your answer as a common fraction.

Solution
$f(-2)+f(-1)+f(0) = \frac{3(-2)-2}{-2-2} + \frac{3(-1)-2}{-1-2} + \frac{3(0)-2}{0-2} = \frac{-8}{-4} + \frac{-5}{-3} + \frac{-2}{-2} = 2 + \frac{5}{3} + 1 = \boxed{\frac{14}{3}}$

Rollout Example
Please integrate natural language reasoning with programs to solve the problem above, and put your final answer within boxed Problem: If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2)+f(-1)+f(0)$? Express your answer as a common fraction. To solve the problem, we need to evaluate the function $f(x) = \frac{3x-2}{x-2}$ at $x = -2$, $x = -1$, and $x = 0$, and then sum the results. Let's break it down step-by-step: 1. Evaluate $f(-2)$:

$$f(-2) = \frac{3(-2)-2}{-2-2} = \frac{-6-2}{-4} = \frac{-8}{-4} = 2$$

2. Evaluate $f(-1)$:

$$f(-1) = \frac{3(-1)-2}{-1-2} = \frac{-3-2}{-3} = \frac{-5}{-3} = \frac{5}{3}$$

3. Evaluate $f(0)$:

$$f(0) = \frac{3(0)-2}{0-2} = \frac{-2}{-2} = 1$$

4. Sum the results:

$$f(-2)+f(-1)+f(0) = 2 + \frac{5}{3} + 1 = 3 + \frac{5}{3} = \frac{9}{3} + \frac{5}{3} = \frac{14}{3}$$

Now, let's confirm this with Python code. "'python This Python script defines a function 'f' that calculates the value of $(3x-2)/(x-2)$. It then evaluates this function at $x = -2$, $x = -1$, and $x = 0$ to find their respective values. The sum of these three values is printed as the final result. "' "'output 4.666666666666667 "' The output from the code is approximately 4.6667, which corresponds to the fraction $\frac{14}{3}$. Thus, the value of $f(-2)+f(-1)+f(0)$ is $\boxed{\frac{14}{3}}$.

D MSARL DUAL-AGENT OPTIMIZATION

Algorithm 1 MSARL: Dual-Agent Optimization via Grouped Rollouts

Input: the reasoning agent π_{reason}, the tool agent π_{tool}, the given question set Q, the ground truth GT, a reward system $\mathcal{R}$
Initialize: reasoning policy parameters θ_{reason}, tool policy parameters θ_{tool}.
Hyperparameters: learning rates of reasoning and tool policies $\alpha_{\text{reason}}, \alpha_{\text{tool}}$; number of reasoning trajectories per question m, number of tool interpretations per trajectory n, maximum tool calls C, PPO clip value ϵ.
for each training iteration **do**
    Sample $\{q_j\} \subset Q$.
    the reasoning and tool datasets:
$D_{\text{reason}} \leftarrow \emptyset$, $D_{\text{tool}} \leftarrow \emptyset$.
    // — Phase 1: Data Generation —
    for each $q \in \{q_j\}$ **do**
        Sample $\{\tau_k\}_{k=1}^{m} \sim \pi_{\text{reason}}(\cdot|q)$.
        for each τ_k **do**
            Let $(t, o)_{\text{last}} \in \tau_k$.
            Sample $\{i_k^{(j)}\}_{j=1}^{n} \sim \pi_{\text{tool}}(\cdot|(t, o)_{\text{last}})$.
            For each $i_k^{(j)}$,
generate $O_k^{(j)} \leftarrow \pi_{\text{reason}}(\tau_k, i_k^{(j)})$.
            Store $(\tau_k, \{(i_k^{(j)}, O_k^{(j)})\}_{j=1}^{n})$.
        end for
    end for
    // — Phase 2: Advantage Estimation —
    for each $(\tau_k, \{(i_k^{(j)}, O_k^{(j)})\}_{j=1}^{n})$ **do**
        $R_k^{(j)} \leftarrow \mathcal{R}(O_k^{(j)}, GT)$.
        $\mu_k \leftarrow \frac{1}{n}\sum_j R_k^{(j)}$; $\sigma_k \leftarrow \sqrt{\frac{1}{n}\sum_j (R_k^{(j)} - \mu_k)^2}$.
        for $j = 1$ to n **do**
            $A_k^{(j)} \leftarrow (R_k^{(j)} - \mu_k)/(\sigma_k + \eta)$.
            $D_{\text{tool}} \leftarrow D_{\text{tool}} \cup \{(i_k^{(j)}, A_k^{(j)})\}$.
        end for
        $\bar{A}_k \leftarrow \frac{1}{n}\sum_j A_k^{(j)}$.
        $D_{\text{reason}} \leftarrow D_{\text{reason}} \cup \{(\tau_k, \bar{A}_k)\}$.
    end for
    // — Phase 3: Policy Update —
    $\theta_{\text{tool}} \leftarrow \theta_{\text{tool}} + \alpha_{\text{tool}} \nabla_{\theta_{\text{tool}}} \mathcal{J}(\theta_{\text{tool}}|D_{\text{tool}})$.
    $\theta_{\text{reason}} \leftarrow \theta_{\text{reason}} + \alpha_{\text{reason}} \nabla_{\theta_{\text{reason}}} \mathcal{J}(\theta_{\text{reason}}|D_{\text{reason}})$.
end for

E STATEMENT ON THE USE OF LARGE LANGUAGE MODELS (LLMS)

In the preparation of this paper, we utilized a Large Language Model (LLM) as a writing assistant to help with language polishing, grammar checking, and text optimization. The use of the LLM was strictly limited to improving the paper's readability and clarity of expression, and was not used to generate any research content, core arguments, or data.

We hereby declare that all research originality, core ideas, experimental methods, results, and conclusions in this paper were developed and finalized by the authors independently. The authors take full responsibility for all aspects of the submission and guarantee its truthfulness and accuracy.