rSIM: Incentivizing Reasoning Capabilities of LLMs via Reinforced Strategy Injection

Anonymous ACL submission

Abstract

Large language models (LLMs) are post-trained through reinforcement learning (RL) to evolve into Reasoning Language Models (RLMs), where the hallmark of this advanced reasoning is "aha" moments when they start to perform *strategies*, such as self-reflection and deep thinking, within chain of thoughts (CoTs). Motivated by this, this paper proposes a novel reinforced strategy injection mechanism (*rSIM*), that enables any LLM to become an RLM by employing a small planner to guide the LLM's CoT through the adaptive injection of reasoning strategies. To achieve this, the planner (leader agent) is jointly trained with an LLM (follower agent) using multi-agent RL (MARL), based on a leader-follower framework and straightforward rule-based rewards. Experimental results show that *rSIM* enables Qwen2.5-0.5B to become an RLM and significantly outperform Qwen2.5-14B. Moreover, the planner is generalizable: it only needs to be trained once and can be applied as a plug-in to substantially improve the reasoning capabilities of existing LLMs. In addition, the planner supports continual learning across various tasks, allowing its planning abilities to gradually improve and generalize to a wider range of problems.

1 Introduction

Large language models (LLMs) have been enhanced with advanced reasoning capabilities, evolving into Reasoning Language Models (RLMs) (Besta et al., 2025) that solve problems through step-by-step reasoning, commonly referred to as chain-of-thought (CoT) (Wei et al., 2022). A key advancement in RLMs is their ability to integrate reasoning *strategies*, such as self-reflection, decomposition, and deliberative thinking, into the CoT process, contributing to improved problem-solving accuracy.

Existing literature (Trung et al., 2024; Havrilla et al., 2024), especially the recent Group Relative Policy Optimization (GRPO) (Guo et al., 2025), primarily post-trains LLMs to evolve into RLMs purely through reinforcement learning (RL) algorithms. A hallmark of this evolution is the emergence of the "aha moment". It presents that LLMs start to perform *strategies* such as self-reflection within CoTs. We empirically found that evolved RLMs perform better when their reasoning strategies appear more frequently and in more appropriate positions. In contrast, LLMs, especially smaller ones, often lack the basic capacity to execute these strategies. Our findings further show that RL-based post-training cannot transform such models into capable RLMs.

Therefore, this paper proposes a reinforced strategy injection mechanism (*rSIM*), which enables any LLM, in particular those as small as 0.5B, to evolve into an RLM with minimal or even no additional training. To achieve this, *rSIM* introduces only an auxiliary planner that, at each reasoning step of an LLM, adaptively selects an appropriate strategy from a predefined set such as *self-reflection*, *decomposition*, and others, and injects it into the chain of thought (CoT) to guide the next step generation. Specifically, the *rSIM* offers the following four key contributions:

- Injecting reasoning strategies adaptively into the CoT process of any LLM, including small and large ones, via a planner enables the LLM to directly gain the advanced reasoning ability like RLMs.

- Training the planner and LLM jointly as two agents under multi-agent RL (MARL) with the leader-follower algorithm (Gerstgrasser and Parkes, 2023) reinforces the planner's ability to inject strategies.

- Planners are pluggable, meaning that a planner trained on one task can be directly integrated

with another LLM to enhance its reasoning ability on similar tasks.

- Planners support continual learning, as a planner in our *rSIM* can be continuously trained across tasks to enhance its planning ability over a broader range of problems.

Our results across seven datasets covering mathematics, multi-task reasoning, and code generation verify the benefits of *rSIM*. First, even small LLMs such as Qwen2.5-0.5B, when jointly trained with a planner (Qwen2.5-0.5B), can evolve into an RLM, achieving accuracy on par with Qwen2.5-14B on `MATH` (Hendrycks et al., 2021). Second, using the trained planner as a plugin enables another LLM to outperform larger models by a significant margin without any additional training. Third, a planner trained on mathematics can be continuously fine-tuned on coding tasks such as `CodeAlpaca-20k` (Chaudhary, 2023) to further guide an LLM like Qwen2.5-0.5B, achieving 17% higher accuracy on code generation.

2 Related Work

Data distillation. Knowledge distilled from large language models (LLMs) can be transferred to smaller models to enhance their performance (Xu et al., 2024). In particular, (Guo et al., 2025) verifies that larger LLMs can transfer their step-by-step reasoning abilities to smaller models by distilling chain-of-thought (CoT) (Wei et al., 2022) samples, where LLM-generated reasoning traces serve as additional fine-tuning data. Fine-tuning with teacher-generated CoT outputs (Magister et al., 2022; Yu et al., 2024; Dai et al., 2024), rationalizations (Li et al., 2023a), specialized reasoning skills (Liao et al., 2025), CoT and Program of Thought (PoT) (Chenglin et al., 2024), or even incorrect CoT samples (Huang et al., 2022; Hosseini et al., 2024) can significantly improve the reasoning abilities of smaller models. In this paper, using the planner from our DPR framework as a plugin to improve the reasoning of smaller models can be viewed as transferring a human-level planning chain to them.

Reinforcement learning. Reinforcement learning (RL) (Thrun and Littman, 2000) has been widely applied to decision-making tasks, as demonstrated by AlphaGo (Silver et al., 2016) and AlphaZero (Silver et al., 2017). RLHF (Ouyang et al., 2022) first leveraged PPO (Schulman et al., 2017) to align models with human preferences. ReFT (Trung et al., 2024) pioneered the use of RL as a fine-tuning paradigm to enhance LLM reasoning performance. Building on this progress, DeepSeek-R1-Zero (Guo et al., 2025) made a breakthrough by demonstrating that self-verification, reflection, and the ability to generate long CoTs in LLMs can be incentivized purely through RL, specifically using GRPO (Shao et al., 2024). This enables smaller base LLMs, such as 3B and 7B models, to be trained directly to achieve reasoning performance comparable to stronger models. However, as noted by (Havrilla et al., 2024), models are inherently constrained in their ability to explore CoT solutions beyond their existing capabilities. Consequently, weaker base models as small as 0.5B fail to benefit from RL training and lag behind in reasoning performance.

Multi-agent LLMs. Building on the development of using a single LLM as a planning or decision-making agent, multi-agent LLMs (Li et al., 2024) based on MARL (Lowe et al., 2017), where multiple language models collaborate, have achieved significant progress in complex problem-solving (Guo et al., 2024; Li et al., 2023b; Zhang et al., 2024). As highlighted by (Li et al., 2024), this structure has been successfully applied to practical tasks (Hong et al., 2023; Qian et al., 2023; Mandi et al., 2024). To enhance performance, SOCRATIC (Shridhar et al., 2023) trains a combination of two small distilled models to perform CoT reasoning in LLMs. In contrast to our DPR framework, SOCRATIC still relies on distilling the abilities of large models into smaller ones. Meanwhile, ReAct (Yao et al., 2023) enables LLMs to generate both reasoning traces and task-specific actions in an interleaved manner. Additionally, CORY (Ma et al., 2024) fine-tunes LLMs as two autonomous agents, a pioneer and an observer, which leads to superior performance compared to standard PPO. However, our work is the first to decouple planning from the reasoning process and to build a two-agent system under the MARL framework that enables any LLM to benefit from the trained planner agent.

3 Preliminary and Motivation

3.1 RLMs via Reinforcement Learning

Given a question q, the reasoning language model (RLM), parameterized by $\boldsymbol{\theta}$, generates a sequence of thoughts, denoted as $\boldsymbol{o} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n]$,

where each $\boldsymbol{z}_i$ with $i \in [1, \ldots, n]$ is a textual description of the thought at the i-th reasoning step. The predicted solution $\widetilde{y}$ is extracted from $\boldsymbol{z}_n$ and compared with the ground truth y. In the reinforcement learning (RL) framework, represented as $[\mathcal{S}, \mathcal{A}, \pi_{\boldsymbol{\theta}}, \mathcal{R}]$, the state $\boldsymbol{s} \in \mathcal{S}$ corresponds to the tokens generated so far, while the action $a \in \mathcal{A}$ is the next token sampled from the policy $a \sim \pi_{\boldsymbol{\theta}}(a|s)$. The corresponding reward is denoted as $r \in \mathcal{R}$. To optimize the policy model $\boldsymbol{\theta}$, the RL algorithm aims to maximize the expected cumulative reward, which is formulated as follows:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}, a \in \boldsymbol{o} \sim \pi_{\boldsymbol{\theta}_{old}}} \left[\frac{\pi_{\boldsymbol{\theta}}(a|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}_{old}}(a|\boldsymbol{s})} A^{\pi_{\boldsymbol{\theta}_{old}}}(\boldsymbol{s}, a) \right]$$

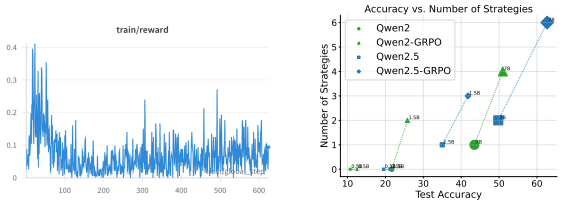
where $\boldsymbol{\theta}_{old}$ represents the most recent policy model. The advantage function $A(\cdot)$ estimates how much better an action is compared to the expected return. To balance bias and variance, we employ Generalized Advantage Estimation (GAE), formulated as $A^{\pi_{\boldsymbol{\theta}_{old}}}(s_t, a_t) = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}$ where $\delta_t = r_t + \gamma V^{\pi_{\boldsymbol{\theta}_{old}}}(s_{t+1}) - V^{\pi_{\boldsymbol{\theta}_{old}}}(s_t)$, where t denotes the token index, and $\gamma \in [0, 1]$ is the discount factor. In general, to mitigate the discrepancy between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_{old}$, methods such as Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Group Relative Policy Optimization (GRPO) (Shao et al., 2024) incorporate a KL penalty term.

As demonstrated by GRPO, the policy model $\boldsymbol{\theta}$ is trained using rule-based rewards, where the rewards encode human-defined rules to guide the model in adhering to these rules while improving reasoning accuracy.

3.2 LLMs Without Inherent Reasoning Strategies Show Limited Improvement

Using a pure RL algorithm like GRPO, a base model is optimized to achieve an "aha" moment — learning to exhibit more reasoning strategies, such as self-reflection and deep thinking, during training, leading to a sudden increase of model response length and ultimately to high accuracy. We argue that (1) when the base LLM is capable of performing reasoning strategies, reinforcement learning (RL) algorithms can optimize it to apply more strategies effectively during reasoning, and (2) there exists a positive correlation between the number of strategies and the accuracy.

To prove this, we first define a set of the most commonly used *strategies*: *self-reflection, decomposition, deliberative thinking, validation, summarization, prioritization, continuation, sub-planning, and termination*. Following subsection A.2, we then count the number of times these strategies appear in the CoTs generated by the base LLMs and post-trained models.



(a) GRPO-based training failure on Qwen2.5-0.5B

(b) Accuracy vs. number of strategies across LLMs

Figure 1: (a) Shows the absence of the "aha" moment in Qwen2.5-0.5B on the MATH dataset (Hendrycks et al., 2021). (b) Compares performance of LLMs before and after training with GRPO. By detecting key words, we count the average number of reasoning strategies, as presented in Figure 2, that are used in answering each question.

As supporting evidence, we train LLMs of sizes 0.5B, 1.5B, and 7B from both Qwen2 and Qwen2.5 on the MATH dataset using the GRPO. First, as shown in Figure 1, when Qwen2.5-0.5B is used as the base model, the total reward initially increases to around 0.3 but then abruptly drops to 0 under GRPO training. Second, from Figure 1b, we observe that base models such as Qwen2-0.5B and Qwen2.5-0.5B, which lack inherent reasoning strategies (i.e., strategy count is 0), cannot be trained with RL to gain reasoning intelligence and show only limited improvement in accuracy. In contrast, models such as 1.5B and 7B, which demonstrate inherent reasoning strategies (with strategy counts greater than 0), can be further optimized through RL to apply more strategies during reasoning. More importantly, we observe that as the number of strategies increases, model accuracy also improves.

Interestingly, our observation **partially aligns** with the findings of Yue et al. (2025), which shows that RL is unable to endow base models with new reasoning capabilities. Our work offers a deeper investigation into **reasoning strategies** and tackles this limitation via a planning injection mechanism.

4 Methodology

Motivated by our observations in Subsection 3.2, we introduce reinforced strategy injection mech-

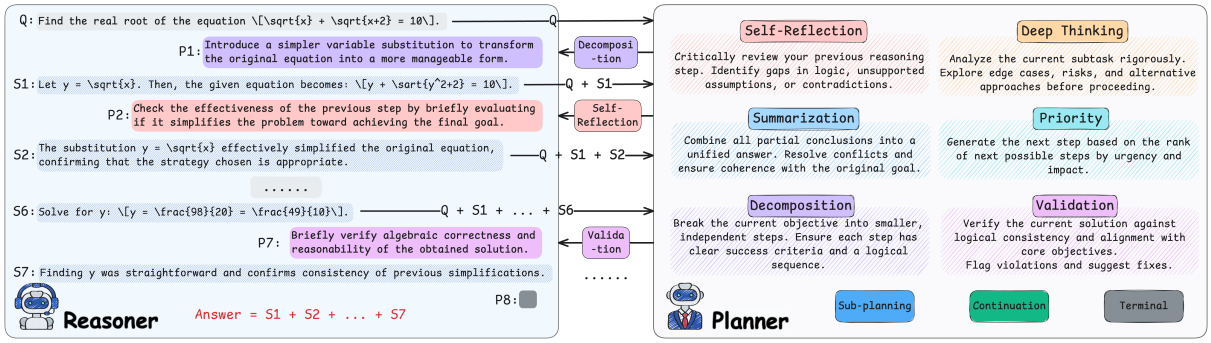


Figure 2: Illustration of the cooperative pipeline between the *rSIM* planner (leader) and the LLM (reasoner/follower). The planner receives the question and the current reasoning steps, and selects one of nine strategies to inject into the reasoning process to guide the reasoner in generating the next step. This demo is based on a question from the MATH dataset.

anism (*rSIM*), which allows a planner, implemented as a small-sized LLM, to guide another LLM by adaptively providing strategy instruction during CoTs. Through injection, the planner can incorporate rich auxiliary knowledge and prior information, encoded as explicit reasoning *strategies*, to enable the LLM to effectively and directly acquire advanced reasoning capabilities and evolve into an RLM.

4.1 Training Objective

The planner is designed to guide the reasoner at each reasoning step by selecting one strategy from a predefined set of nine human-designed strategies (Figure 2), which encode core reasoning strategies that LLMs should follow. These strategies, expressed as prompts, help bridge the gap in weak LLMs that inherently lack such capabilities, enabling meaningful improvement during post-training with reinforcement learning. In this framework, the base LLM serves as the reasoner, responsible for generating the next reasoning step based on the planner's selected strategy. This collaboration is naturally modeled as a leader-follower paradigm (Gerstgrasser and Parkes, 2023) within a multi-agent RL system. The planner, acting as the leader, takes an action $\boldsymbol{a}^p \sim \pi^p_{\boldsymbol{\phi}}(\boldsymbol{a}^p|\boldsymbol{s})$, where $\pi^p_{\boldsymbol{\phi}}$ is the policy parameterized by $\boldsymbol{\phi}$, $\boldsymbol{a}^p$ is a strategy sampled from the action space $\mathcal{A}^p$, and $\boldsymbol{s}^p \in \mathcal{S}^p$ includes the question and previous reasoning steps. The reasoner, as the follower, then takes an action $a \sim \pi_{\boldsymbol{\theta}}(a|\boldsymbol{s}, \boldsymbol{a}^p)$, where a is the next token and $\boldsymbol{s}$ denotes all tokens generated thus far. Eventually, we define the planner's reward as $R^p = R_{acc} + R_{terminal} + R_{penalty}$ and the reasoner's reward as $R = R_{acc} + R_{format} + R_{follow}$, where $R_{terminal} = 1$ if the final plan is the 'Terminal' strategy, and -1 otherwise; $R_{penalty} = -$ (ratio of the most frequently selected strategy); R_{follow} is the ratio of reasoning steps that follow the given plan; and R_{acc} and R_{format} are the accuracy and format rewards as defined in GRPO (Shao et al., 2024).

As presented in Figure 2, to answer each question, the two agents interact with each other for n rounds, where n corresponds to the number of reasoning steps required to produce the final answer, leading to $\boldsymbol{o}^{dpr} = [\boldsymbol{p}_1, \boldsymbol{z}_1, \boldsymbol{p}_2, \boldsymbol{z}_2, \ldots, \boldsymbol{p}_n, \boldsymbol{z}_n]$, where $\boldsymbol{p}_n$ is plan selected by the planner for the n-th reasoning step. In the context of reinforcement learning, we define two advantage functions: $A^{\pi^p_{\boldsymbol{\phi}}}(\boldsymbol{s}_t, \boldsymbol{a}^p_t)$ for the planner and $A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t, \boldsymbol{a}^p)$ for the reasoner. Here, t denotes the index of the generated token. For the planner, we assume that all tokens within a single reasoning step share the same advantage, which is set equal to the plan-level reward. Therefore, we have the following objective $\mathcal{J}_{\boldsymbol{o}^{dpr}}$:

$$\frac{1}{|\boldsymbol{o}^{dpr}|}\sum_{t=1}^{|\boldsymbol{o}^{dpr}|}\left[\lambda\cdot\left(\frac{\pi^p_{\boldsymbol{\phi}}(\boldsymbol{a}^p_t|\boldsymbol{s}_t)}{\pi^p_{\boldsymbol{\phi}_{\text{old}}}(\boldsymbol{a}^p_t|\boldsymbol{s}_t)}\right)\cdot A^{\pi^p_{\boldsymbol{\phi}}}(\boldsymbol{s}_t, \boldsymbol{a}^p_t)\right.$$
$$\left.+(1-\lambda)\cdot\left(\frac{\pi_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t, \boldsymbol{a}^p_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|\boldsymbol{s}_t, \boldsymbol{a}^p_t)}\right)\cdot A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t, \boldsymbol{a}^p_t)\right]$$

where $\boldsymbol{o}^{dpr} \sim (\pi^p_{\boldsymbol{\phi}_{old}}, \pi_{\boldsymbol{\theta}_{old}})$, and we set $A^{\pi^p_{\boldsymbol{\phi}}}(\boldsymbol{s}_t, \boldsymbol{a}^p_t) = A^{\pi^p_{\boldsymbol{\phi}}}(\boldsymbol{s}_i, \boldsymbol{a}^p_i)$ if the t-th token belongs to the reasoning step $\boldsymbol{z}_i$.

We follow the GRPO to define the final objective as $\mathbb{E}\left[\frac{1}{G}\sum_{j=1}^{G}\mathcal{J}_{\boldsymbol{o}^{dpr}_j}\right]$, where $A^{\pi^p_{\boldsymbol{\phi}}}(\boldsymbol{s}_t, \boldsymbol{a}^p_t) = \frac{R^p_j - \text{mean}\left(\{R^p_j\}_{j=1}^{G}\right)}{\text{std}\left(\{R^p_j\}_{j=1}^{G}\right)}$.

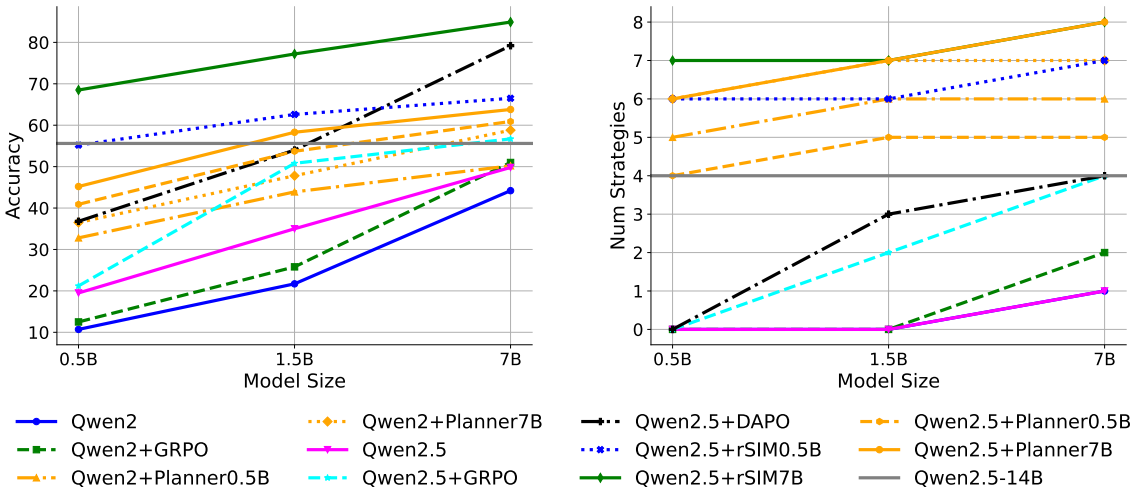


Figure 3: Accuracy of Qwen2.5 models (0.5B, 1.5B, 7B) on the `MATH` dataset under various settings. "+GRPO" and "+DAPO" indicate training with GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025), respectively. "+*rSIM*" denotes joint training with a Qwen2.5 planner, with the number indicating planner size. "+Planner" refers to using trained planners (0.5B, 7B) in plugin mode, derived from the `MATH` dataset (see Figure 5).

4.2 A Two-Stage Training Scheme

We observe that simultaneously and equally optimizing the policies of both agents often leads to several issues: (1) conflicting policy updates, where the gradients of the two agents may "pull" in opposing directions; (2) credit assignment ambiguity, as it becomes unclear whether success or failure is due to the leader's plan or the follower's execution; and (3) competing exploration versus exploitation, where joint exploration may result in catastrophic miscoordination.

Therefore, to enable effective training, we propose a two-stage scheme in which the first stage prioritizes the policy optimization of the planner, while the second stage shifts focus to optimizing the policy of the reasoner, i.e., the base model. Specifically, we ensure a stable training process by adjusting the weighting parameter λ across the two stages. In the first stage, we set $\lambda = 0.7$ to emphasize planner optimization, and in the second stage, we reduce it to $\lambda = 0.3$ to prioritize the optimization of the reasoner. To keep the main paper concise, further implementation details are provided in the appendix.

5 Experiments

Datasets. Experiments are performed on seven datasets from the HuggingFace website: 1). `MATH` (Hendrycks et al., 2021), `GSM8K` (Cobbe et al., 2021), `AIME2024` for mathematics, 2). `MMLU-Pro` (Wang et al., 2024), TheoremQA (Chen et al., 2023) for multi-task reasoning, 3) `CodeAlpaca-20k` (Chaudhary, 2023) and `HumanEval` for code generation. Further experimental setup is provided in subsection A.5 of the Appendix.

Training Settings. Our experiments uses a range of LLMs, including Qwen2 models at 0.5B, 1.5B, and 7B scales, as well as Qwen2.5 models at 0.5B, 1.5B, 7B, and 14B scales. In addition, we incorporate Open-o1 and Deepseek-R1 (Guo et al., 2025) as base reasoners, which are paired with the trained planner used as a plugin. We use a batch size of 16 with gradient accumulation set to 4. For the GRPO-related hyperparameters, we configure the temperature at 0.9, the maximum prompt and completion lengths at 1024 each, G at 16, and the KL coefficient (beta) at 0.04. The learning rates for the 0.5B, 1.5B, 3B, 7B, and 14B models are 2e-5, 1e-5, 8e-6, 5e-6, and 2e-6, respectively. These settings remain consistent for both the planner and the reasoner. We employ the AdamW optimizer with a cosine learning rate scheduler. During evaluation, we set the planner's temperature to 0 and the reasoner's temperature to 0.3.

Baselines. We compare *rSIM* with the recent GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025), which are two advanced reinforcement learning methods for language models. In addition, we include two baseline approaches: Plan-and-Solve (PS+) prompting (Wang et al., 2023) and Planner Prompting, which is typically implemented within the ReAct framework (Yao et al., 2023). Specifically, the "w/ prompting" condition in our experiments refers to Planner Prompting, in

Table 1: Performance of Llama series models with *rSIM* on four datasets such as TheoremQA, along with comparisons to the Plan-and-Solve (PS+) Prompting method (Wang et al., 2023) and Planner Prompting baseline. When the reasoner (in the Models column) collaborates with a planner, the planner's size is indicated instead of using 'No'. When a model name such as Qwen2.5 is specified in the 'Planner' column, we perform cross-model evaluation, where Llama serves as the reasoner and Qwen2.5 as the planner. When the "Planner" column specifies 'plug-in', the planner is used off-the-shelf without training to guide the reasoner. Moreover, the row containing 'w/ prompting' indicates that we directly prompt an LLM to act as the planner. The '-' and '×' denote unconverged training and missing results due to untrainable models, respectively. 0 means no strategy used by the LLM.

Llama3.2 Models	**Planner**	MATH		GSM8K		MMLU-Pro		TheoremQA	
		Score	**#Strategy**	**Score**	**#Strategy**	**Score**	**#Strategy**	**Score**	**#Strategy**
1B w/ ZeroCoT		30.6	0	44.4	0	21.2	0	13.7	0
1B w/ PS+ [4]		28.2	0	43.7	0	19.4	0	12.2	0
1B w/ prompting	3B	27.4	7	42.6	3	16.8	8	6.6	6
1B w/ prompting	70B	33.3	5	46.9	3	22	6	14.3	5
3B w/ ZeroCoT		48	0	77.7	0	30.1	0	20.7	0
3B w/ PS+ [4]		47.5	0	77.7	0	30	0	18.6	0
3B w/ prompting	3B	46.4	7	77.1	5	28.5	7	19.9	8
3B w/ prompting	70B	55.5	7	81.8	4	31.8	5	22.8	6
1B w/ GRPO		-	0	-	0	×	0	×	0
1B w/ *rSIM*	1B	57	3	83.9	1	30.8	4	20.9	3
1B w/ *rSIM*	3B	61.5	4	86.3	3	33	6	25	6
1B w/ *rSIM*	Qwen2.5-1.5B	59.1	4	84.4	1	31.8	4	24.2	5
Llama3.3 70B w/ ZeroCoT		77	0	90.5	0	68.9	0	32.3	0
Llama3.3 70B w/ PS [4]		78.3	0	90.9	0	70	0	32	0
Llama3.3 70B w/ Prompting	3B	79.1	7	90.5	4	68.9	7	31.9	8
Llama3.3 70B w/ Prompting	70B	84	6	92.9	4	71.5	4	38.6	5
Llama3.3 70B	1B plug-in	83.2	3	91.7	1	71.8	6	39	5
Llama3.3 70B	3B plug-in	86.3	4	92.1	2	72.7	5	41.8	6
Llama3.3 70B	Qwen2.5-1.5B	83.7	4	92	2	72.3	6	40.7	6

which a large language model (LLM) is directly instructed to act as a planner and generate a step-by-step reasoning strategy (ones defined in our *rSIM*) for each stage of the output generation process.

Metrics. The primary evaluation metric is accuracy, defined as the percentage of correct answers. Additionally, to assess the planner's effectiveness as a guidance mechanism, we measure the average number of strategies applied per problem.

5.1 Main Results

With *rSIM*, any LLM, especially smaller ones, can be trained to convergence, achieving dramatic improvements in reasoning performance, as shown by the training curve in Figure 5 and the high problem-solving accuracy in Figure 3. Specifically, for the base Qwen2.5 models in sizes of 0.5B, 1.5B, and 7B, jointly training with a planner, either Qwen2.5-0.5B or Qwen2.5-7B, under the results in significant accuracy gains over the base models, even surpassing stronger base models. For example, all models trained with *rSIM* (tagged as "+*rSIM*" in Figure 3) outperform their GRPO-trained counterparts ("Qwen2.5+GRPO"). Moreover, the base Qwen2.5-0.5+*rSIM*0.5B model, which employs a 0.5B planner, notably outperforms the Qwen2.5-7B model. Overall, all Qwen2.5+*rSIM*7B models across different base sizes achieve a new state-of-the-art (SOTA) performance. Moreover, as shown in Table **??**, the planner introduced by our *rSIM* does not significantly increase the token cost when facilitating reasoning in a single LLM.

By comparing the left and right sub-figures of Figure 3, we observe that after training under *rSIM*, every base model learns to execute a variety of human-level reasoning strategies as defined by the nine options in Figure 2. In particular, even weaker base models, such as Qwen2.5-0.5B, which initially showed no discernible strategy, can be guided by the planner to employ six or seven strategies during reasoning. These results also reveal a positive correlation between accuracy and the number of strategies used.

More importantly, as shown in Table 1, applying *rSIM* to other types of LLMs, such as a series of Llama models, still yields consistent performance improvements due to the strategy injection provided by the planner. For example, when Llama3.2-1B is used as the reasoner and paired with a Llama3.2-1B planner, the system achieves accuracy close to that of Llama3.3-70B. This suggests that the planner in our DPR framework is

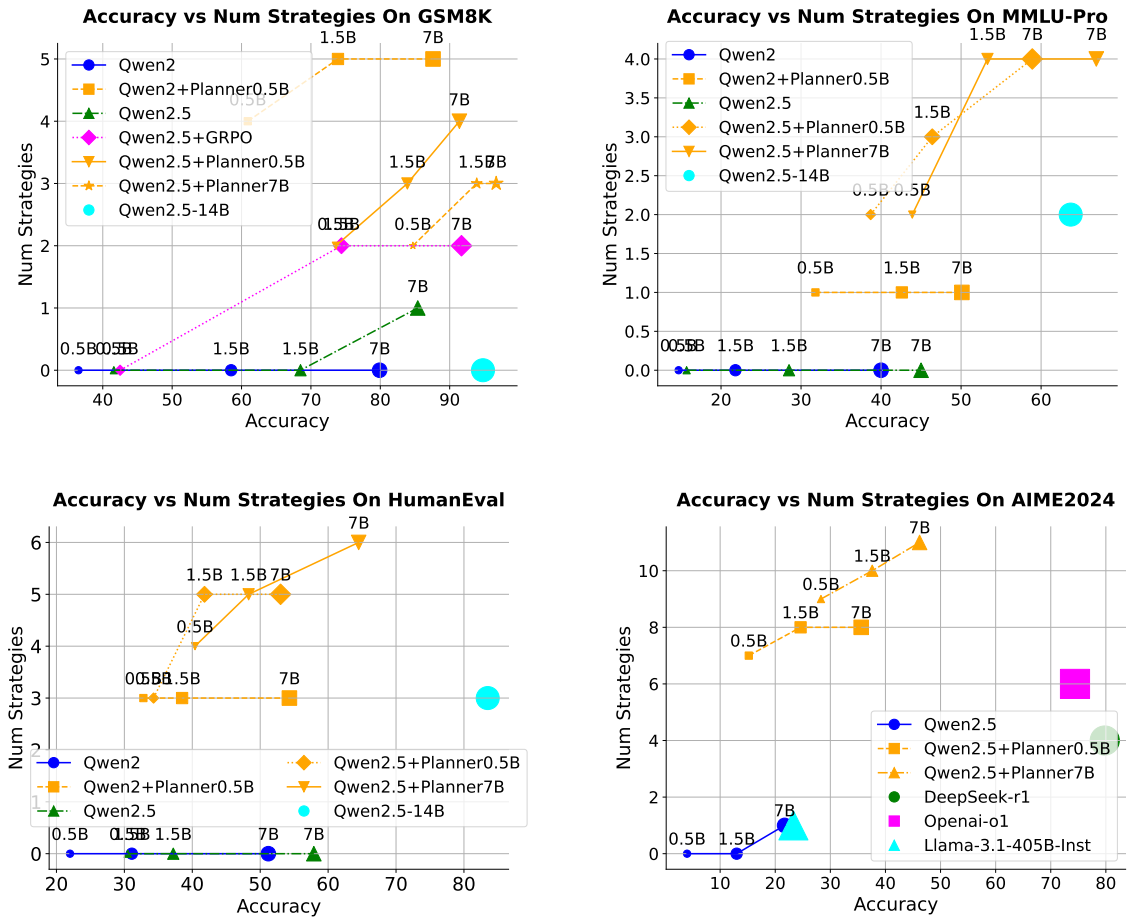


Figure 4: Illustration of accuracy for models of various sizes across different datasets, using the trained *rSIM* planner as a plugin. The terms with "+Planner" indicate that the base model collaborates with a trained planner during reasoning. For `HumanEval`, we use the planner trained on the `CodeAlpaca-20k` dataset, while the planners for all other benchmarks are trained on the `MATH` dataset, as shown in Figure 5.

transferable across different LLMs, although the degree of improvement may depend on the base model used for planning.

In addition, we present cross-model evaluation results in Table 1, where the trained planner (Qwen2.5-1.5B) is used to assist Llama3.2-1B and Llama3.3-70B by injecting step-wise strategies. The reasoning performance of the Llama models shows an improvement, indicating that the *rSIM* planner possesses cross-model generalization capability.

5.2 Evaluation of the Pluggable Planner

As shown in Figures 3 and 4, which report the performance of base LLMs on five datasets, the trained planner can be used as a plugin to directly introduce human-level reasoning strategies **without any additional post-training**. In the `MATH` dataset (Figure 3), Qwen2.5+Planner models achieve significantly higher accuracy than both the base Qwen2.5 models and those post-trained with GRPO. A similar trend is observed across all four datasets in Figure 4: once integrated with the pre-trained planner, base models show substantial accuracy gains and often outperform larger counterparts. For instance, Qwen2.5-1.5B+Planner consistently surpasses the 7B models, and Qwen2.5-7B+Planner achieves SOTA accuracy, even outperforming more powerful models such as Llama-3.1-405B-Inst in the final subfigure of Figure 3. More importantly, as shown by the number of strategies used during problem-solving across all datasets, the trained planner enables any LLM to incorporate effective reasoning strategies into the reasoning process. This effect is especially evident in the challenging `AIME2024` dataset, where the planner enables LLMs to apply human-level strategies more than eight times per problem, resulting in a significant improvement in accuracy.

Reasoner	Planner (*MATH*)	MATH	GSM8K	HumanEval	MMLU_Pro	AIME2024
0.5B	0.5B	-0.8	+0	+17	+0.4	+0
	7B	+0	+1.1	+22.3	+0.9	+0
7B	0.5B	+1.2	+1.8	+24.4	+0	+0
	7B	+0	+0.6	+22.8	+0	+0

Table 2: Performance gain of the off-the-shelf planner after continued training on the coding task `CodeAlpaca-20k` under the *rSIM* . With the enhanced planner used as a plugin, the reasoner models (listed in the "reasoner" column: Qwen2.5-0.5B and Qwen2.5-7B) address problems across five datasets.

These results demonstrate that once trained, planners can be directly reused to enhance the performance of LLMs on other tasks. For instance, planners such as Qwen2.5-0.5B and Qwen2.5-7B, trained with the simple Qwen2.5-0.5B reasoner on the `MATH` dataset, work adaptively with different LLMs across various datasets. This opens a new direction where, instead of fine-tuning LLMs for every task, we can integrate high-intelligence modules such as the planner in *rSIM* to directly boost their reasoning capabilities. In doing so, we shift the focus from optimizing a coupled planning and reasoning process to developing an advanced planner that can collaborate with any LLM.

Similarly, when using Llama series models as base LLMs, the trained planner can serve as a plug-in module to directly inject human-level reasoning strategies. As shown in Table 1, without any additional post-training, Llama3.2-1B, when guided by strategies provided by the Llama3.2-1B or Llama3.2-3B planner, achieves consistent and significant accuracy improvements across all corresponding datasets.

5.3 Evaluating the Continuous Learning Capability of the Planner

Table 2 shows that the planner can be continuously trained to achieve improved performance on all tasks while preserving its effectiveness on previously seen tasks. Specifically, the off-the-shelf planner, originally optimized on the `MATH` dataset as shown in Figure 5, is further trained with Qwen2.5-0.5B (serving as the reasoner) using the `CodeAlpaca-20k` dataset under the *rSIM* to obtain a coding-enhanced planner. This enhanced planner is then used as a plugin with the reasoner models (0.5B and 7B of Qwen2.5) to address questions from five diverse datasets. Table 2 reports the performance gains relative to the off-the-shelf `MATH` planner. Overall, aside from a small drop of 0.8 in accuracy on `MATH`, the performance of the enhanced planner is either maintained or improved across all tasks. In particular, the improvement on `HumanEval` is substantial, with gains ranging from 17% to 24.4%. This demonstrates that continued training allows the planner to better guide coding-related reasoning using human-level strategies. Such a significant advantage highlights the practical value of the planner, as it can be continuously trained with a small reasoner on mixed-task datasets to improve its **capabilities in a cumulative way**. It is worth noting that the coding-enhanced planner does not show any accuracy gain on `AIME2024`.

6 Ablation Study

In Table 1 and Table 6, we compare the planner of *rSIM* with baseline prompting methods, including PS+ prompting (Wang et al., 2023) and direct planner prompting. From our results, we observe that these prompting-based approaches generally require a powerful language model, such as Qwen2.5 14B, to yield even limited improvements. When the model is smaller, such as LLaMA3.2 3B or Qwen2.5 0.5B, accuracy often decreases, demonstrating the limited reliability and scalability of prompting for plan generation. In addition, as shown in Table 5 and Figure 6, we examine the impact of different strategies on reasoning performance across various datasets. The results show that self-reflection is consistently important, while other strategies' effectiveness varies across tasks.

7 Concluding Remarks

In this paper, we proposed the reinforced strategy injection mechanism (*rSIM*), which enables any large language model (LLM), including small ones like Qwen2.5-0.5B, to become an advanced reasoning language model (RLM). *rSIM* trains a planner with a reasoner jointly via multi-agent reinforcement learning using a leader-follower algorithm. The planner learns to select the best strategy from a set of nine human-designed options to guide the reasoner step by step. Experiments showed that *rSIM* significantly improves reasoning accuracy across multiple tasks. Importantly, the planner can be used as a plugin without additional training, enabling any LLM to gain advanced reasoning capabilities immediately. The planner also supports continual learning, enabling its reasoning guidance to improve continuously across diverse tasks.

References

Maciej Besta, Julia Barth, Eric Schreiber, Ales Kubicek, Afonso Catarino, Robert Gerstenberger, Piotr Nyczyk, Patrick Iff, Yueling Li, Sam Houliston, and 1 others. 2025. Reasoning language models: A blueprint. *arXiv preprint arXiv:2501.11223*.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. Theoremqa: A theorem-driven question answering dataset. In *Proc. Conference on Empirical Methods in Natural Language Processing*.

Li Chenglin, Qianglong Chen, Liangyue Li, Caiyu Wang, Feng Tao, Yicheng Li, Zulong Chen, and Yin Zhang. 2024. Mixed distillation helps smaller language models reason better. In *Findings of the Association for Computational Linguistics*, pages 1673–1690.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Chengwei Dai, Kun Li, Wei Zhou, and Songlin Hu. 2024. Improve student's reasoning generalizability through cascading decomposed cots distillation. In *Conference on Empirical Methods in Natural Language Processing*.

Matthias Gerstgrasser and David C Parkes. 2023. Oracles & followers: Stackelberg equilibria in deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 11213–11236. PMLR.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. In *Proc International Joint Conference on Artificial Intelligence*, pages 8048–8057.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024. Teaching large language models to reason with reinforcement learning. In *International Conference on Machine Learning*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. In *Conference on Language Modeling*.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve. *Advances in neural information processing systems*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. *Transactions on Machine Learning Research*.

Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. 2023a. Symbolic chain-of-thought distillation: Small models can also" think" step-by-step. In *The Association for Computational Linguistics*.

Yuan Li, Yixuan Zhang, and Lichao Sun. 2023b. Metaagents: Simulating interactions of human behaviors for llm-based task-oriented coordination via collaborative generative agents. *arXiv preprint arXiv:2310.06500*.

Huanxuan Liao, Shizhu He, Yao Xu, Yuanzhe Zhang, Kang Liu, and Jun Zhao. 2025. Neural-symbolic collaborative distillation: Advancing small language models for complex reasoning tasks. In *Association for the Advancement of Artificial Intelligence*.

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

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Liu Boyin, Xiaolin Ai, Yanyan Liang, and Min Chen. 2024. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 37:15497–15525.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2022. Teaching small language models to reason. In *The Association for Computational Linguistics*.

Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. Roco: Dialectic multi-robot collaboration with large language models. In *IEEE International Conference on Robotics and Automation*, pages 286–299.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 6(3).

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. 2023. Distilling reasoning capabilities into smaller language models. In *Findings of the Association for Computational Linguistics*, pages 7059–7073.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and 1 others. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, and 1 others. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *nature*.

Sebastian Thrun and Michael L Littman. 2000. Reinforcement learning: An introduction. *AI Magazine*, 21(1):103–103.

Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. In *Proc Annual Meeting of the Association for Computational Linguistics*, pages 7601–7614.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proc. Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 2609–2634.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*.

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

Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. 2024. Distilling system 2 into system 1. In *Workshop on Neural Information Processing Systems*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237.

A Implementation Details

This section presents the design, training, and evaluation details of our proposed *rSIM*.

A.1 Finetuning the Reasoner LLM toward Generating Step-wise Reasoning

We first ensure that the reasoner LLM generates the reasoning process in a step-by-step format. To accomplish this, we adopt the mechanism proposed by (Lightman et al., 2023), finetuning the model to use '\n\n' to clearly separate each reasoning step in the problem-solving process. Specifically, we construct a prompt consisting of five chain-of-thought examples formatted such that each step is separated by '\n \n'. Using this prompt, we let the reasoner generate reasoning processes randomly on the `MATH` dataset. To avoid introducing dataset-specific solution information into the model, we select only those generated samples that conform to the desired format but contain incorrect solutions. Ultimately, we obtain 1,000 such samples and fine-tune the reasoner on these samples for one epoch. This process ensures that the reasoning generated by the model consistently follows the intended step-by-step formatting.

A.2 Counting the Number of Strategies

To determine how many strategies are used by the LLM during reasoningas shown in Fig. 1(B), Fig. 4, and Fig. 5we adopt a keyword-matching approach based on strategy names and their synonyms. Specifically, we construct a list of candidate keywords for each of the seven main strategies: *self-reflection*, *decomposition*, *deep thinking*, *validation*, *summarization*, *prioritization*, and *sub-planning*, as follows:

- Self-Reflection: review, revisit, reflect, reevaluate, rethink, reexamine, reassess, reconsider, analyze, assess, validate, critique, inspect, examine, audit, diagnose, cross-check.
- Decomposition: decompose, break down, divide, split, separate, segment, partition, dissect, analyze, unfold, unwrap, reduce, map out, organize, structure.
- Deep Thinking: contemplate, deliberate, reflect, ponder, mull over, reason, deduce, infer, evaluate, scrutinize, meditate, analyze, consider, investigate, explore.
- Validation: validate, verify, confirm, check, test, justify, prove, cross-check, ensure, affirm, support, substantiate, corroborate, authenticate, evaluate.
- Summarization: summarize, recap, restate, paraphrase, condense, outline, highlight, abstract, generalize, simplify, extract, distill, encapsulate, conclude, report.
- Prioritization: prioritize, rank, order, select, choose, emphasize, highlight, focus on, weigh, assess, sort, filter, arrange, allocate, favor.
- Sub-planning: plan, outline, design, strategize, organize, arrange, map out, formulate, structure, prepare, coordinate, blueprint, set up.

Therefore, once a reasoning step is completed, indicated by the generation of '\n\n', we identify keywords within that sequence. If a match is found, we increment the count for the corresponding strategy by one.

A.3 Structure of the Planner

The planner is a simple decoder-only LLM equipped with an action heada linear layer that outputs a 9-dimensional vector corresponding to the number of available strategies. The planner takes as input the question and the current reasoning steps. The hidden state of the final token in the sequence is fed into the action head to produce a strategy priority vector. The strategy with the highest score is then selected to guide the reasoner's next reasoning step. For example, when using Qwen2.5-0.5 as the planner, the action head is implemented as a fully connected layer with shape 896×9. In this case, the hidden vector of the final token in the input sequence serves as the input to the action head for strategy selection.

A.4 Interactive Sampling Mechanism

Different from GRPO (Shao et al., 2024), which generates G samples for each question by directly forwarding the question through the LLM G times, our multi-agent framework adopts an interactive sampling mechanism in which the planner agent and the reasoner agent of *rSIM* interact throughout the reasoning process to generate each sample. We follow the notation introduced in Subsections 3.1 and 4.1, and thus present the detailed procedure in Algorithm Table 1. It is important to note that when the planner collaborates with an LLM

Algorithm 1: Interactive Sampling Mechanism

Input: Question q, the reasoner policy $\pi_{\boldsymbol{\theta}}$ with its hyperparameters, such as the temperature, the planner policy $\pi_{\boldsymbol{\phi}}^{p}$ with its hyperparameters, and the number of generations G.

Output: Generated G samples.

Sample G first strategies from the planner $\left\{a_{1,g}^{p}\right\}_{g=1}^{G} \sim \pi_{\boldsymbol{\phi}}^{p}\left(\cdot|q\right)$

Begin parallel sampling:

for *any* $a_1^p \in \left\{a_{1,g}^{p}\right\}_{g=1}^{G}$ **do**

  $a_{trace}^{p} \leftarrow a_1^p, \quad \boldsymbol{o}^{dpr} = [], \quad \boldsymbol{a}^p = [a_1^p], \; n \leftarrow 1$

  while a_{trace}^{p} *is not* ***Terminal*** **do**

    Autoregressively Decode with $\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}\left(\cdot|q, \boldsymbol{o}^{dpr}, a_{trace}^{p}\right)$ until '\n\n' is generated

    Collect the sequence of decoded $\boldsymbol{a}$ as the reasoning step $\boldsymbol{p}_n, \boldsymbol{s}_n$

    Append $\boldsymbol{o}^{dpr} \leftarrow \left[\boldsymbol{o}^{dpr}, \boldsymbol{p}_n, \boldsymbol{s}_n\right]$

    Select strategy action $a^p \sim \pi_{\boldsymbol{\phi}}^{p}\left(\cdot|q, \boldsymbol{o}^{dpr}\right)$

    Set $a_{trace}^{p} \leftarrow a^p$, Append $\boldsymbol{a}^p \leftarrow [\boldsymbol{a}^p, a_{trace}^{p}]$

    Set $n \leftarrow n + 1$

  end

end

Return $\left\{\boldsymbol{o}_{1...G}^{dpr}\right\}, \left\{\boldsymbol{a}_{1...G}^{p}\right\}$

Algorithm 2: Policy Optimization of Multi-Agent Framework

Input: q, reasoner policy $\pi_{\boldsymbol{\theta}}$, planner policy $\pi_{\boldsymbol{\phi}}^{p}$, dataset $\mathcal{D}$, N.

Output: Optimized $\boldsymbol{\theta}, \boldsymbol{\phi}$.

for $e = 1, \ldots, E$ **do**

  Set reference models $\pi_{\boldsymbol{\theta}_{ref}} \leftarrow \pi_{\boldsymbol{\theta}}$, $\pi_{\boldsymbol{\phi}_{ref}} \leftarrow \pi_{\boldsymbol{\phi}}$

  $lambda \leftarrow 0.7$

  for *step* $1, \ldots, M$ **do**

    Sample a batch samples $\mathcal{D}_b$ from $\mathcal{D}$

    Update old policy models $\pi_{\boldsymbol{\theta}_{old}} \leftarrow \pi_{\boldsymbol{\theta}}, \quad \pi_{\boldsymbol{\phi}_{old}} \leftarrow \pi_{\boldsymbol{\phi}}$

    Perform *Interactive Sampling Mechanism* to generate $\left\{\boldsymbol{o}_{1...G}^{dpr}\right\}, \left\{\boldsymbol{a}_{1...G}^{p}\right\}$ for each question in $\mathcal{D}_b$

    Compute rule-based rewards $\left\{R_j^p\right\}_{j=1}^{G}$ and $\{R_j\}_{j=1}^{G}$ for each $\boldsymbol{o}_j^{dpr}, \boldsymbol{a}_j^p$

    Compute advantage $A^{\pi_{\boldsymbol{\phi}}^{p}}(\boldsymbol{s}_t, \boldsymbol{a}_t^p)$for the t-th token of each $j \in G$ through group relative advantage estimation.

    $lambda \leftarrow 0.3$ iff. $step < N$

    Update policy models $\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\phi}}^{p}$ by maximizing the training objective (Equation 1)

  end

end

for problem solving, the step-by-step reasoning is generated following the same procedure.

A.5 Training and Evaluation Details

Our experiments are performed on two types of tasks:

Mathematics. `MATH` refers to the Hugging Face dataset 'DigitalLearningGmbH/MATH-lighteval', which contains 7500 samples for training and 5000 for testing. `GSM8K` refers to 'openai/gsm8k' on Hugging Face, which contains 7470 samples for training and 1320 for testing. `AIME2024` refers to 'Maxwell-Jia/AIME_2024' on Hugging Face, which contains 30 samples for testing.

Coding. `CodeAlpaca-20k` is the Hugging Face dataset 'HuggingFaceH4/CodeAlpaca_20K', which contains 18000 samples for training. `HumanEval` refers to 'openai/openai_humaneval' on Hugging Face, which contains 164 samples for testing.

Multi-task. `MMLU-Pro` refers to 'TIGER-Lab/MMLU-Pro' on Hugging Face, which contains 70 samples for validation. `TheoremQA` refers to the 'TIGER-Lab/TheoremQA' dataset on Hugging Face, which contains 800 samples for test. We only use te these datasets for the evaluation.

Evaluation setup. We set the maximum prompt length to 2,048 tokens and the maximum generation length to 1,024 tokens for all models. During evaluation, we adopt the zero-shot setting with a temperature of 0, and report pass@1 (accuracy) using stochastic decoding. When using the planner as a plugin for evaluation, we strictly follow

the *Interactive Sampling Mechanism* described in Algorithm Table 1.

We perform **three modes** of evaluations in the main paper. First, we finetune the LLM with our *rSim* on the `MATH` and `GSM8K`, leading to the `MATH` and `GSM8K` planners, respectively. Besides, we also finetune the LLM with our *rSim* on the `CodeAlpaca-20k` to obtain the coding planner.

To evaluate planner performance, we first assess the planners directly on the test sets of `MATH` and `GSM8K`, yielding Figure 3 (for Qwen models) and the first two columns of Table 1 (for Llama models).

Next, we employ the planner trained on `MATH` (referred to as the `MATH` planner) as a plug-in module for reasoning on `MMLU-Pro` and `TheoremQA`, producing the last two columns of Table 1. We further apply the same `MATH` planner as a plug-in for the reasoner on `GSM8K`, `MMLU-Pro`, and `AIME2024`, resulting in the results shown in Figure 4. Separately, we use the coding-specific planner, trained on `CodeAlpaca-20k`, as a plug-in for `HumanEval`, also reported in Figure 4.

Using our *rSim* framework, we **continuously** fine-tune the `MATH` planner on the `CodeAlpaca-20k` dataset to investigate whether the planner can acquire new planning strategies for coding tasks while preserving its original performance on mathematical reasoning. The resulting planner, obtained by fine-tuning the `MATH` planner on `CodeAlpaca-20k`, is then evaluated across mathematical, coding, and multi-task benchmarks, with results presented in Table 2.

By combining these settings and evaluations, we present a comprehensive analysis of our *rSim* framework, highlighting its ability not only to directly improve the reasoning performance of LLMs, especially smaller ones, but also to support three key capabilities: **Plugin Planner**, **Cross-Task Transfer Analysis**, and **Continual Learning**.

A.6 Learning Details and Rewards

The planner only needs to select one of the strategies from the given set, similar to a classification procedure (the planning is not free generation and thus does not need SFT cold start). As RL is a trial-and-error process, when the planner selects a bad strategy leading to a wrong answer from the reasoner, a negative reward is assigned to reduce the planner's probability of selecting that strategy.

The planner and reasoner interact contextually, yet without output masking during training. At each step, the planner selects a strategy given the past strategies selected, while the reasoner is the normally LLM decoding which generates a reasoning step given the planner strategy injected in this step as well as the full context of reasoner and planner outputs thus far. We let the planner perform strategy selection only after the reasoner completes one step of reasoning. Please refer to subsection fsubsec:stepwise of the Appendix for details of how to enable an explicit step-wise reasoning of the reasoner and identify one step.

The **reward design** is presented in subsection 4.1, where we do not introduce any novel reward design, but rather follow the most commonly used reward designs in RL fine-tuning literature, e.g., GRPO. Similar to the literature, the reward for reasoning includes a formatting and accuracy reward and advantage comparisons among choices. In other words, we use standard RL rewards for reasoning problems. Rather, our novel contribution lies in enabling effective cooperation between planner and reasoner via a conceptually simple and clean strategy injection protocol, and making it work on small models (even as small as 0.5B) with a MARL framework.

Following the literature, here is a detailed explanation of the reward for both the planner and reasoner, including three parts: accuracy reward, format reward, and guidance reward. The accuracy reward is computed based on the output (whether the final answer is correct). The format reward, commonly seen in GRPO, is to ensure that the planner and reasoner follow the desired format, while the guidance reward is used as an auxiliary to make the agent follow human priors.

A.7 Traces, Rollouts and Learning

We define a rollout as a trace $\tau = \{(a_1^p, \boldsymbol{s}_1), \ldots, (a_T^p, \boldsymbol{s}_T)\}$ generated by the planner policy $\pi_{\boldsymbol{\phi}}^p$ and the reasoner policy $\pi_{\boldsymbol{\theta}}$. Initialization proceeds with planner and reasoner contexts that are isolated to avoid leakage.

Initialization. For each question q, the planner first samples an initial strategy $a_1^p \sim \pi_{\boldsymbol{\phi}}^p(\cdot \mid q)$ using only the question context. The corresponding reasoner context is initialized as $\mathcal{C}_1^r = [q,\, g(a_1^p)]$, where $g(\cdot)$ maps a strategy ID to a short guidance tag, ensuring that the reasoner receives only the current strategy signal and no planner internals.

Masked contexts. At step $n \geq 2$, the planner context is $\mathcal{C}_n^p = [q,\ f_{obs}(\boldsymbol{s}_1), \ldots, f_{obs}(\boldsymbol{s}_{n-1})]$, where $f_{obs}(\cdot)$ summarizes completed reasoning steps without exposing ground-truth answers, logits, or future plans. The planner never observes the reasoner's partial tokens for the current step. The reasoner context is $\mathcal{C}_n^r = [q,\ \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n-1},\ g(a_n^p)]$; it does not receive planner scores, future strategies, or raw planner prompts beyond the minimal tag $g(a_n^p)$. Prior planner outputs are excluded from $\mathcal{C}_n^r$ to prevent strategy leakage across steps.

Rollout and termination. Given $\mathcal{C}_n^r$, the reasoner decodes a step $\boldsymbol{s}_n$ until the step delimiter is produced, forming the observation sequence $\boldsymbol{o}^{dpr} = [\boldsymbol{p}_1, \boldsymbol{s}_1, \ldots, \boldsymbol{p}_n, \boldsymbol{s}_n]$. The planner then selects a_{n+1}^p from $\mathcal{A}$. The rollout terminates when a terminal strategy is chosen or an end-of-sequence is reached.

Learning. We optimize both policies by maximizing $J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\phi}}^p}\big[\sum_{t=1}^{T} R_t + \lambda R_t^p\big]$, where R_t and R_t^p are step-level rewards for the reasoner and planner, respectively. Advantages are computed at token level for the reasoner and via group-relative estimation for the planner, consistent with Algorithm Table 2. Masked contexts $\mathcal{C}_n^p, \mathcal{C}_n^r$ are enforced during both training and evaluation to ensure isolation and prevent information leakage.

A.8 Algorithm Details

Algorithm Table 2 presents how to jointly optimize the policies of the two agents. During the *Interactive Sampling Mechanism*, the temperatures for the planner and reasoner are set to be 0.9. We set the maximum prompt length to 2,048 tokens and the maximum generation length to 1,024 tokens for all models. **It is worth noting that in the training objective (Equation 2), we include only $\left\{\boldsymbol{o}_{1\ldots G}^{dpr}\right\}$ for simplicity, although the full formulation should involve both $\left\{\boldsymbol{o}_{1\ldots G}^{dpr}\right\}, \{\boldsymbol{a}_{1\ldots G}^{p}\}$. We should note that for tokens generated guided by the action of the planner, this action reward is assigned to each token's reward.**

B Additional Results

As shown in Fig. 5, we train the Qwen2.5-0.5B model, initially lacking high-level reasoning capabilities, using *rSIM* with planners implemented by both Qwen2.5-0.5B and Qwen2.5-7B models. With the two-stage training scheme, total rewards steadily increase until convergence, stabilizing around 2.8 for the 0.5B planner and 3.5 for the 7B planner. Crucially, the planner and reasoner policy models effectively learn to generate and follow step-wise strategies, respectively, resulting in improved reasoning performance. Additionally, the 7B planner achieves higher rewards more rapidly than the 0.5B planner. Consistent training and evaluation trends confirm effective policy optimization. Therefore, we conclude that **by decoupling planning from reasoning, we can introduce human priors in the form of integrated reasoning strategies into any LLM that lacks them, significantly enhancing reasoning intelligence through our multi-agent RL framework**.

Table 3 presents a summary comparison between our *rSim* approach, supervised fine-tuning (SFT), and advanced RL-based methods, namely GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025). The results reveal a key advantage of *rSim*: while SFT and RL-based methods struggle to improve performance on small language models such as Qwen2.5-0.5B because these models inherently

Table 3: Comparison of our *rSIM* method with supervised fine-tuning (SFT) and RL-based methods on the `MATH` and `HumanEval` datasets. The Qwen2.5 models are used. "–" indicates non-convergent training.

Reasoner	Planner	`MATH` (%)	# Strategies	`HumanEval` (%)	# Strategies
GRPO	0.5B	–	–	0.0	–
DAPO	0.5B	36.8	0	31.5	0
SFT	1.5B	40.0	0	37.9	0
GRPO	1.5B	50.8	2	35.7	0
DAPO	1.5B	53.8	3	41.3	2
rSIM	**0.5B**	**55.2**	**6**	**47.5**	**4**

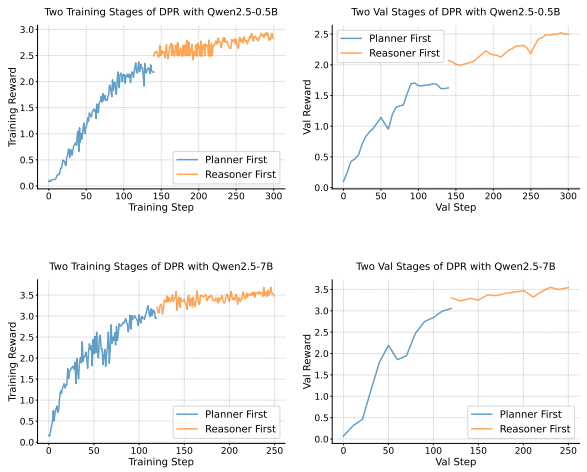


Figure 5: Training and evaluation (Eval) curves of the *rSIM* on the `MATH` dataset, using either the Qwen2.5-0.5B or Qwen2.5-7B model as the planner, paired with the Qwen2.5-0.5B model as the reasoner.

Table 4: Evaluation of the average generation token cost across different methods on MATH and TheoremQA. We report both the average and standard deviation (mean ± std) of the total tokens used per question, including tokens used for prompting the LLMs and those generated by the models. This evaluation is conducted on two challenging datasets: MATH and TheoremQA.

Methods	Planner	MATH	TheoremQA
0.5B w/ ZeroCoT	No	221.6 ± 172.5	250.3 ± 110.2
14B w/ ZeroCoT	No	261.8 ± 192.2	308.7 ± 137.5
0.5B w/ PS+ [4]	No	327.5 ± 176.7	367.5 ± 153.6
0.5B w/ Prompting	14B	815.2 ± 356.7	993.4 ± 390.5
0.5B w/ *rSIM*	7B	780.3 ± 210.9	800.7 ± 230.2
7B w/ ZeroCoT	No	246.9 ± 189.5	291.2 ± 160.8
7B w/ PS [4]	No	357.2 ± 200.9	390.6 ± 190
7B w/ Prompting	14B	934.5 ± 390.2	1103 ± 487.5
7B w/ *rSIM*	7B	900.6 ± 350.8	970.2 ± 427.1

lack the capacity to acquire complex reasoning strategies through standard training, our method bypasses this limitation by directly injecting structured planning strategies through an external planner. As a result, *rSim* achieves substantial accuracy gains even on compact models.

For **Compute Efficiency Analysis**, Table 4 reports the average generation token cost across methods. Relative to planner prompting with a 14B planner, *rSIM* reduces tokens per question while preserving accuracy: for the 0.5B reasoner, by about 4.3% on MATH (815.2 to 780.3) and 19.4% on TheoremQA (993.4 to 800.7); for the 7B reasoner, by 3.6% (934.5 to 900.6) and 12.0% (1103 to 970.2), respectively. It also stabilizes compute with smaller variance (e.g., 356.7 to 210.9 on MATH and 390.5 to 230.2 on TheoremQA for the 0.5B reasoner). While *rSIM* uses more tokens than lightweight baselines such as ZeroCoT or PS, these extra tokens reflect explicit planning and yield higher accuracy, offering a favorable efficiency-performance trade-off.

B.1 Additional Ablation Study

Table 5 presents the importance of different strategies in improving the reasoning accuracy of Qwen2.5-7B planner in different datasets. Due to time constraints, we do not re-adjust the strategy set for fine-tuning the LLMs across various tasks, nor do we directly compute statistical metrics for each strategy. Instead, we evaluate the importance of each strategy by iteratively masking out one strategy at a time during the evaluation phase. This allows us to observe how the performance is affected when a specific strategy is not considered by the planner. Specifically, when a strategy is masked out, the planner selects the alternative strategy with the second-highest score.

C Limitations

The limitations of our proposed *rSIM* can be categorized into three main aspects: (1) the human-defined action space of the planner is not continuously optimizable and (2) the planner exhibits an imbalanced preference over strategies.

First, the planner's action space is defined solely based on human understanding of the task. As a result, its effectiveness, diversity, and generalization depend heavily on expert priors and cannot be enhanced by the planner itself during reasoning. Without broader exploration during trainingparticularly in our multi-agent reinforcement learning settingthe *rSIM*'s performance may significantly degrade when necessary strategies are absent from the planner's action space. Furthermore, since each strategy is encoded as a short prompt describing the guidance, the quality and robustness of these prompts directly affect the reasoner's reasoning generation. Misleading, ambiguous, or overly narrow prompts can harm the reasoning process. Unfortunately, such issues are common in human-crafted prompts, including biases and lack of applicability across tasks. More importantly, we currently lack a mechanism to dynamically update or expand the planner's action space. Although this challenge is shared by many reinforcement learning-based approacheswhere any change to the action space invalidates the underlying policyit is particularly critical in our context. Enhancing the strategy space is necessary to improve the generality of LLM-based reasoning frameworks, which is essential for their practical deployment at scale.

Second, we count the total number of strategies used by both the planner (Qwen2.5-0.5B) and the reasoner (Qwen2.5-0.5B) during problem-solving across different datasets. As shown in Fig. 6, the planner consistently emphasizes the use of *Self-Reflection* across all four datasets, while placing less focus on *Validation*, *Prioritization*, and *Sub-Planning*. Notably, in the MATH dataset, the second most frequently used strategy is *Decomposition*, which is also true for the coding task HumanEval. In addition, for the complex mathematical problems in AIME2024, *Decomposition* again shows a high usage rate, indicating that

Table 5: Evaluation of the importance of different strategies on reasoning performance when Qwen2.5-0.5 serves as the reasoner and Qwen2.5-7B acts as the planner during inference. In each row labeled with a strategy name, we remove the corresponding strategy from the planner's option set to evaluate its impact on reasoning performance.

Dataset/Strategy	full	self-reection	deep thinking	decomposition	summarization	validation	prioritization	sub-planning
MATH	45.2	36.9	41.3	42.8	43.7	44.3	44.6	44.8
HumanEval	40.2	32.3	39	35.4	38.4	39	39.6	35.4
MMLU Pro	43.9	31.3	38.7	34.5	39.6	41.8	42.7	44.1
TheoremQA	38.7	30	35.3	32.8	38.4	38.6	37.1	36.9

Table 6: Performance of our *rSIM* on datasets such as TheoremQA, along with comparisons to the Plan-and-Solve (PS) Prompting method (Wang et al., 2023) and Planner Prompting baseline. We use Qwen2.5 models in sizes 0.5B, 7B, and 14B. The format and experimental setting of this table are consistent with those in Table 1.

Methods	**Planner**	MATH		MMLU-Pro		TheoremQA	
		Score	**#Strategy**	**Score**	**#Strategy**	**Score**	**#Strategy**
0.5B w/ ZeroCoT	No	19.5	0	15.7	0	9.5	0
0.5B w/ PS+ [4]	No	17.2	0	13	0	8	0
0.5B w/ Prompting	7B	21.6	7	15.9	8	9.6	8
0.5B w/ Prompting	14B	26.3	6	18.6	6	10	5
14B w/ ZeroCoT	No	55.6	0	51.2	0	43	0
14B w/ PS+ [4]	No	57.1	0	52.3	0	43.4	0
14B w/ Prompting	7B	56.8	6	53.5	6	43.8	7
14B w/ Prompting	14B	60	5	57	5	47.4	5
0.5B w/ *rSIM*	0.5B	40.9	4	38.7	2	34.1	3
0.5B w/ *rSIM*	7B	45.2	6	43.9	2	38.7	4
7B w/ ZeroCoT	No	49.8	0	40	0	36	0
7B w/ PS [4]	No	49.6	0	39.2	0	34.9	0
7B w/ Prompting	7B	51	6	40.2	8	36.4	7
7B w/ Prompting	14B	56.8	5	46.7	6	41.8	5
7B w/ *rSIM*	0.5B	60.9	5	58.9	4	48.3	6
7B w/ *rSIM*	7B	63.8	8	66.9	4	53	5

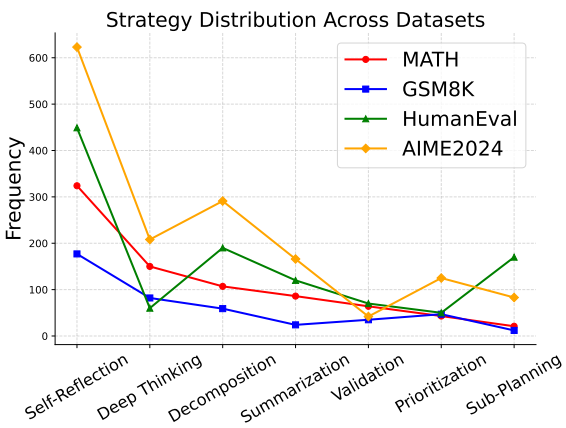


Figure 6: Illustration of how many strategies are used in solving 100 questions from MATH, GSM8K, HumanEval, and AIME2024.

breaking down the problem during reasoning is crucial for arriving at the correct solution. Therefore, since strategies are not used in a balanced manner during the problem-solving of questions across different tasks, the planner may be unable to comprehensively explore diverse solution paths to reach a reliable answer.

D More Discussions

Recent advancements in Large Language Model (LLM) reasoning have largely focused on prompting strategies that elicit better reasoning paths, such as Chain-of-Thought (CoT), Tree-of-Thoughts (ToT), and Plan-and-Solve (PS) (Wang et al., 2023). While effective, these methods typically rely on static, human-designed prompts (e.g., "Let's think step by step" or "Plan first, then solve") that are applied uniformly across all instances. In contrast, *rSIM* introduces a

dynamic, learnable componentthe Plannerwhich treats reasoning strategy selection as a decision-making problem. Unlike static prompting, *rSIM* decouples the *planning* of reasoning strategies from the *execution* of reasoning steps. This allows the system to adaptively select the most effective high-level strategy (e.g., Decomposition vs. Self-Reflection) for a specific problem instance, rather than relying on a one-size-fits-all prompt. Furthermore, compared to inference-time search methods which perform expensive exploration at test time, *rSIM* amortizes the cost of search by training a policy (the Planner) to predict optimal strategies, thereby offering a more compute-efficient alternative to heavy test-time scaling while surpassing the performance of static prompting baselines.

The mechanism of injecting strategies via a planner in *rSIM* can be viewed through the lens of meta-learning, or "learning to learn." In our framework, the Planner acts as a meta-learner that optimizes the learning process of the Reasoner. Instead of directly learning the mapping from input to output (as in standard supervised fine-tuning), the Planner learns a meta-policy over a set of cognitive strategies. This aligns with the concept of meta-reasoning, where an agent monitors and regulates its own computational processes. By reinforcing the selection of strategies that lead to correct answers, *rSIM* effectively performs meta-level optimization: it learns *how* to approach a problem (e.g., by breaking it down or verifying steps) rather than just memorizing specific solution patterns. This connection suggests that strategy injection is a form of meta-knowledge transfer, where abstract reasoning patterns are distilled into the Planner's policy and then instantiated by the Reasoner. This perspective opens new avenues for enhancing LLM intelligence, suggesting that future work could focus on expanding the meta-action space (the set of strategies) and employing more advanced meta-learning algorithms to generalize reasoning capabilities across diverse domains.