Policy-Guided Stepwise Action Planning for Controllable LLM Reasoning

Jianpeng Zhou[1], Qisheng Hu[2], Jiahai Wang[1*], Wenya Wang[2*]
[1]School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2]College of Computing and Data Science, Nanyang Technological University, Singapore
zhoujp7@mail2.sysu.edu.cn, qisheng001@e.ntu.edu.sg
wangjiah@mail.sysu.edu.cn, wangwy@ntu.edu.sg

Abstract

Steering large language model (LLM) reasoning via high-level reasoning actions offers a promising approach to improve robustness and interpretability. However, existing action-based paradigms, ranging from training-free prompting to static plan retrieval or prediction, often fail to consistently outperform standard generation because their planners tend to degenerate into repetitive loops or fixed patterns. We propose PG-HAP (Policy-Guided High-Level Action Planning), a lightweight stepwise planner–executor framework that learns to select reasoning actions dynamically while keeping the executor LLM fully frozen. The planner is trained with reinforcement learning to optimize answer correctness. To prevent degeneration, we introduce two targeted mechanisms: (i) an *Action-Dependency Logit Mask* that enforces valid transitions to avoid redundancy, and (ii) an *Action Diversity Reward* that discourages mode collapse by promoting varied action sequences. Across mathematical and commonsense reasoning benchmarks, PG-HAP improves accuracy over strong baselines while producing less redundant, more adaptive trajectories. This demonstrates that learning high-level planning alone can substantially strengthen reasoning without expensive end-to-end model tuning.[1]

1 Introduction

Large language models (LLMs) have demonstrated impressive reasoning capabilities when guided by prompting strategies such as Chain-of-Thought (CoT) (Wei et al., 2022). However, classic approaches (Kojima et al., 2022; Wei et al., 2022; Chen et al., 2023; Zhou et al., 2023) typically rely on fixed or implicitly learned reasoning patterns, offering limited control over how different reasoning strategies are invoked. To improve controllability, recent work introduces explicit high-level reasoning actions (e.g., decomposition, verification, analysis) that structure generation into a sequence of interpretable decisions. This gives rise to a *planning–execution* paradigm, in which reasoning is decoupled into two roles: a *planner* that determines the high-level strategy (i.e., which action to take) and an *executor* that instantiates each action into concrete reasoning content. Existing approaches that leverage such actions can be broadly divided into three categories. First, training-free methods based on prompt engineering (e.g., CoM (Liu et al., 2025)) rely on the executor model's implicit planning ability to select actions step-by-step. Second, retrieval-based methods (e.g., HiAR-ICL (Wu et al., 2024)) retrieve and reuse fixed action templates collected offline. Third, supervised planning methods (e.g., DOTS (Yue et al., 2025)) train an external planner to predict a complete action trajectory in advance, which is then executed by the LLM.

Despite the growing interest in this planning–execution paradigm, existing methods rarely examine how high-level actions should behave to support effective reasoning. In practice, increasing the complexity of planning mechanisms does not reliably translate into stronger reasoning performance compared to simple CoT prompting. We attribute this performance gap to two critical limitations shared by existing approaches, as visualized in Figure 1. **Structural redundancy** in training-free methods: Training-free approaches (e.g., CoM) rely heavily on the executor's implicit planning priors, which often results in unstable reasoning trajectories. As shown in Figure 1(a), such trajectories frequently contain repetitive loops (e.g., Analysis → Analysis) or skip essential steps, producing redundancy rather than meaningful strategic diversity. **Action sequence collapse** in retrieval-based and supervised planners: In contrast, retrieval-based and supervised planning methods (e.g., HiAR-ICL, DOTS) tend to converge to a small set of rigid action tem-

*Corresponding authors.
[1]Code: https://github.com/john1226966735/PG-HAP.

Findings of the Association for Computational Linguistics: ACL 2026, pages 40740–40765

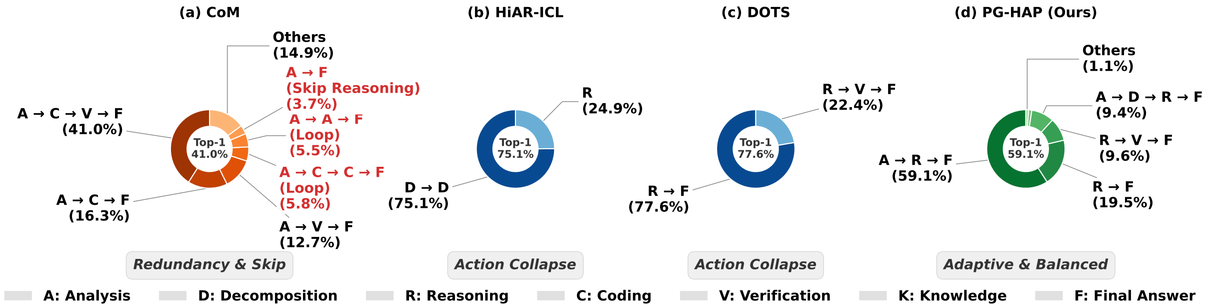

Figure 1: Distribution of top reasoning action sequences on GSM8K. Existing approaches exhibit two distinct failure modes: (1) Structural Redundancy (e.g., CoM, shown in (a)), characterized by repetitive loops and unstable trajectories; and (2) Action Sequence Collapse (e.g., HiAR-ICL, DOTS, shown in (b)(c)), where planners converge to one or two action templates. In contrast, PG-HAP (shown in (d)) yields a balanced distribution, achieving adaptive reasoning strategies while avoiding both redundancy and mode collapse.

plates. As illustrated in Figure 1(b) and (c), the vast majority of instances follow nearly identical action sequences, limiting the planner's ability to adapt to problem-specific requirements.

In this work, we propose Policy-Guided High-Level Action Planning (PG-HAP), a framework that controls a frozen executor LLM through learned, *interleaved* stepwise high-level action selection. Unlike prior approaches that either rely on implicit prompting heuristics or commit to a complete action plan before execution begins, PG-HAP makes each action decision conditioned on the current reasoning state, allowing the planner to adapt its strategy dynamically as intermediate results emerge. Specifically, PG-HAP incorporates two targeted mechanisms to ensure trajectory quality. (1) **Action-Dependency Logit Masking** constrains the planner's action space to logically valid transitions, preventing redundant loops and incoherent step sequences. (2) **Action Diversity Reward** penalizes the overuse of identical action sequences at the batch level, encouraging exploration of multiple viable reasoning strategies rather than convergence to a single template. These mechanisms regularize policy learning, yielding interpretable action policies that are both less repetitive and more adaptive.

We evaluate PG-HAP on multiple mathematical and commonsense reasoning benchmarks. Our results demonstrate that PG-HAP consistently outperforms both training-free and instance-level planning baselines. Further analysis confirms that our approach effectively reduces repetitive behaviors while maintaining inference efficiency.

Our contributions are summarized as follows:

- We provide an empirical analysis of reasoning action sequences in existing action-based methods, revealing systematic patterns of structural redundancy and action sequence collapse.
- We propose PG-HAP, a lightweight planner–executor framework that enables stepwise, controllable selection of high-level reasoning actions for a frozen LLM.
- We introduce simple yet effective training mechanisms, specifically an action-dependency logit mask and a diversity reward, to promote diverse and valid reasoning trajectories.
- We demonstrate the effectiveness of PG-HAP through extensive experiments, showing significant performance gains and improved trajectory quality compared to state-of-the-art baselines.

2 Related Work

Prompting Strategies for Reasoning. Standard approaches to eliciting reasoning in Large Language Models (LLMs) rely on prompting techniques that encourage intermediate generation steps. Methods such as Chain-of-Thought (CoT) (Wei et al., 2022), Program-of-Thought (Chen et al., 2023; Gao et al., 2023), Plan-and-Solve (Wang et al., 2023), and Least-to-Most prompting (Zhou et al., 2023) decompose complex problems into sub-tasks. Further advancements incorporate self-verification or refinement mechanisms (Madaan et al., 2023) to improve robustness. Despite their effectiveness, these methods typically apply a *static* reasoning template across all instances. They lack the flexibility to adapt the reasoning strategy dynamically based on the unique features of various problems, often leading to insufficient reasoning on hard problems or redundant verbosity on simple ones.

High-Level Action Planning. To achieve finer controllability, recent work enables LLMs to utilize explicit high-level reasoning actions (e.g., *decompose*, *verify*) (Qi et al., 2025; Wu et al., 2024; Yue et al., 2025; Liu et al., 2025; Ma et al., 2025). Existing approaches fall into two main categories. Training-free methods, such as Chain of Methodologies (Liu et al., 2025) and Cognitive Tools (Ebouky et al., 2025), rely on the executor LLM's implicit planning capabilities to invoke actions, but often suffer from *structural redundancy*—degenerating into repetitive loops or semantically incoherent action sequences. Search-based approaches, including DOTS (Yue et al., 2025) and HiAR-ICL (Wu et al., 2024), retrieve or optimize action trajectories offline, producing higher-quality plans but functioning as *instance-level* planners that fix the entire sequence before inference and cannot adapt to errors mid-reasoning. In contrast, PG-HAP learns a *stepwise* policy, dynamically adjusting its strategy at each step while explicitly promoting action diversity.

Reinforcement Learning for Reasoning. Reinforcement learning (RL) has emerged as a powerful paradigm for enhancing LLM reasoning. End-to-end approaches, exemplified by recent long-chain reasoning models (e.g., OpenAI o1 (Jaech et al., 2024), DeepSeek-R1 (Guo et al., 2025)) and outcome-supervision methods (Shao et al., 2024; Yu et al., 2025), optimize the entire generation process, implicitly encouraging complex behaviors like backtracking or self-correction. However, these methods typically operate at the token level, requiring substantial training resources and resulting in "black-box" reasoning processes that are difficult to interpret or control explicitly. Concurrently, Plan-Then-Action (Dou et al., 2025) generates a complete high-level plan as a text prefix before execution, optimizing the entire plan–execution sequence end-to-end with GRPO. Our work differs in two key respects: (i) PG-HAP performs *interleaved* stepwise planning where each action decision conditions on the evolving reasoning trace, rather than committing to a full plan upfront; and (ii) the executor remains strictly frozen, so all improvements are attributable to the learned planning policy. More broadly, our work adopts a modular perspective: instead of tuning the massive executor LLM, we apply RL solely to a lightweight planner. This decoupling enables interpretable, controllable action selection and efficient training, achieving the benefits of optimized reasoning without the prohibitive cost of full-model alignment. Importantly, PG-HAP is orthogonal to end-to-end RL approaches—the planner can be deployed on top of an already RL-tuned executor to provide explicit action-level controllability.

3 Method

3.1 Overview

Existing action-based reasoning paradigms often struggle with two distinct failure modes: **structural redundancy**, where a model repeatedly invokes the same action within a single problem-solving episode (e.g., looping *Analysis*), and **action sequence collapse**, where the model converges to a single generic action sequence for all problems regardless of difficulty. To address these challenges, we propose **PG-HAP** (**P**olicy-**G**uided **H**igh-level **A**ction **P**lanning).

The core philosophy of PG-HAP is to decouple reasoning into two distinct, specialized models: a *lightweight planner* that selects high-level reasoning actions in a stepwise manner, and a *frozen executor* that generates the corresponding reasoning content. As illustrated in Figure 2, the framework operates as a Markov Decision Process (MDP). At each time step, the planner encodes the current reasoning state and selects a discrete cognitive action. The executor then instantiates this action into a specific prompt template to generate the next reasoning step. This interaction loop is stabilized by two key mechanisms: *Action-Dependency Logit Masking* to ensure valid transitions, and an *Action Diversity Reward* to mitigate collapse. Crucially, this *interleaved* planning paradigm differs from approaches that generate a complete action plan as a prefix before any execution begins: each action decision in PG-HAP is conditioned on the actual intermediate outputs produced so far, enabling mid-reasoning adaptation that upfront planners cannot perform.

3.2 Problem Formulation

We model the stepwise reasoning process as a Markov Decision Process (MDP). Given a question q, the system generates a reasoning trajectory $\tau = (s_1, a_1, o_1, \ldots, s_T, a_T, o_T)$, where the goal is to optimize the correctness of the final answer.

State Space. The state s_t at time step t encapsulates the question and the complete interaction

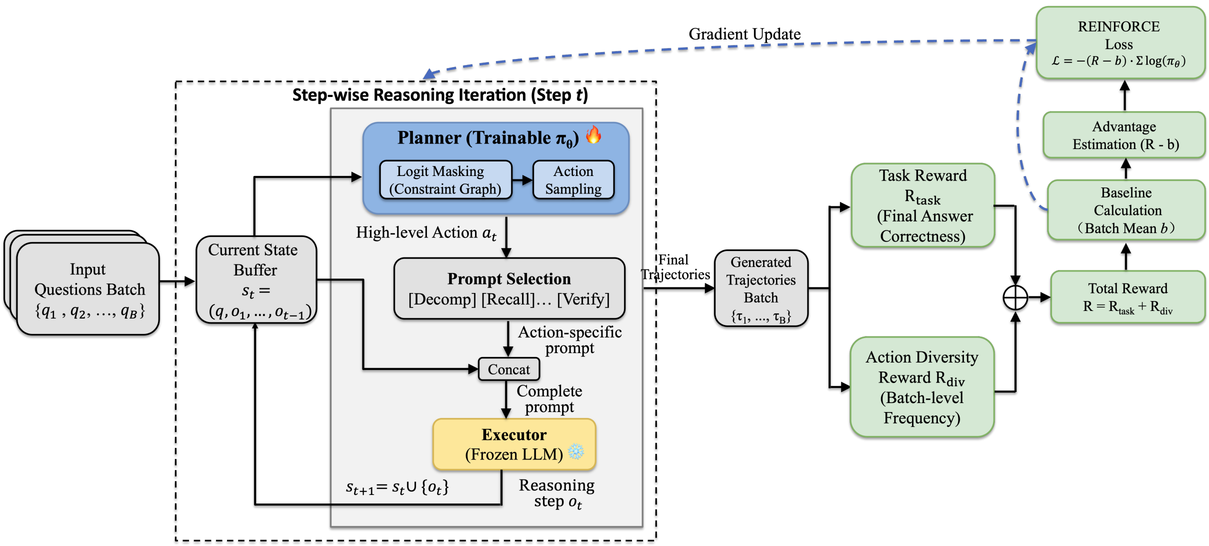

Figure 2: Overview of the PG-HAP framework. The Planner is a standalone lightweight module (frozen Qwen2.5-1.5B backbone + trainable head) that selects high-level actions. The selected action wraps the context with a specific template, guiding the frozen Executor to generate the next reasoning step. Training is regularized by (1) an action-dependency mask to enforce logical transitions and (2) a diversity reward to prevent action collapse.

history up to that point. It is defined as:

$$s_t = [q, a_1, o_1, \dots, a_{t-1}, o_{t-1}],$$

where a_i represents the action taken at step i, and o_i is the reasoning content generated by the executor.

Action Space. The action space $\mathcal{A}$ consists of predefined cognitive operations designed to structure the stepwise reasoning process. Specifically, the fine-grained action set includes: **Problem Analysis**, **Problem Decomposition**, **Direct Reasoning**, **Continue Reasoning**[2], **Knowledge Recall**, **Code Generation**, **Code Refinement**, **Verification**, and **Final Answer**. The used action templates can be found in Appendix J.

Prompt Instantiation. The planner and executor operate in a decoupled manner. At step t, the planner observes s_t and selects a high-level action $a_t \in \mathcal{A}$. This action is then instantiated into a natural language prompt $\mathcal{T}(a_t)$. The frozen executor receives $[s_t; \mathcal{T}(a_t)]$ and generates the next reasoning segment o_t.

3.3 Planner: Lightweight Policy Learning

The planner is instantiated as a distinct, lightweight module explicitly decoupled from the executor. We utilize Qwen2.5-1.5B-Instruct as the semantic encoder. This backbone remains strictly frozen. Given the state s_t, we linearize the interaction history into a single continuous text sequence. Specifically, we concatenate the question q with the chronological sequence of previously selected actions and their corresponding execution results (i.e., $q \to a_1 \to o_1 \to \dots$). This sequence is fed into the backbone to compute the hidden state $h_t \in \mathbb{R}^d$ of the last token. This embedding is then projected by a trainable linear head to predict the next action distribution:

$$P(a_t \mid s_t) = \mathrm{Softmax}(\mathbf{W}h_t + \mathbf{b}).$$

This design effectively leverages the LLM's pre-trained context understanding while optimizing only a lightweight classification layer.

3.3.1 Action-Dependency Logit Masking

To prevent invalid trajectories and ensure logical coherence, we implement an **Action-Dependency Logit Mask** that dynamically constrains the action space based on the current context. The mask is designed around three structural principles: (1) *no degenerate self-loops*—the planner cannot repeat the same action consecutively; (2) *no premature termination*—at least one reasoning step must be generated before the final answer; and (3) *existence of valid multi-step reasoning paths*—every admissible action leads to at least one complete trajectory. For example, *Verification* is only permitted after

[2] Although our method distinguishes between Direct (DR) and Continue (CR) Reasoning for precise control, we group them under the general label "Reasoning (R)" in Figure 1 to align with the coarser action definitions of baseline methods.

Current Action (Symbol)	Valid Next Actions
Analysis (A)	CR, D, C, K
Knowledge Recall (K)	CR, D, C
Decomposition (D)	CR, C
Direct Reasoning (DR)	CR, V, F
Code Generation (C)	CR, CF, F
Continue Reasoning (CR)	V, F
Code Refinement (CF)	CR, F
Verification (V)	CR, V, F
Final Answer (F)	–

Table 1: Action transition constraints enforced by the logit mask. The first column lists the full action name and its symbol; the second column lists the symbols of valid subsequent actions.

a reasoning step because it operates on a derived intermediate result; allowing it directly after *Knowledge Recall* would amount to "verifying a retrieval," which is semantically ill-defined.

As detailed in Table 1, the mask $\mathbf{M}_t$ sets the probabilities of invalid actions to zero:

$$\tilde{\pi}_\theta(a_t \mid s_t) \propto \pi_\theta(a_t \mid s_t) \cdot \mathbf{M}_t.$$

The mechanism handles three scenarios:

Start of Reasoning: If $t = 1$, only valid entry actions (e.g., *Analysis*, *Decomposition*) are allowed.

Stepwise Transition: For intermediate steps, allowed next actions are determined by the transition graph shown in Table 1. For instance, *Verification* is only permitted after reasoning steps.

Termination: If the maximum step limit is reached ($t = T_{\max}$), the mask forces the selection of the *Final Answer* to ensure the trajectory concludes within the computational budget.

3.3.2 Action Diversity Reward

To mitigate *dataset-level action collapse*—where the planner overfits to a few dominant action sequences—we introduce a batch-level **Action Diversity Reward**.

For a given question q, we sample a batch of N trajectories $\mathcal{B}_q = \{\tau_1, \ldots, \tau_N\}$. Let $\mathcal{S}(\tau_i)$ denote the action sequence of τ_i. We compute a diversity score u_i based on the frequency of the sequence within the batch:

$$c_i = \sum_{j=1}^{N} \mathbb{I}[\mathcal{S}(\tau_j) = \mathcal{S}(\tau_i)], \quad u_i = 1 - \frac{c_i - 1}{N - 1},$$

where $\mathbb{I}[\cdot]$ denotes the indicator function. This reward penalizes frequent, redundant sequences while encouraging unique reasoning paths.

Optimization. The final reward for trajectory τ_i combines task correctness ($R_{\text{task}} \in \{0, 1\}$) and diversity:

$$R_i = R_{\text{task}}(\tau_i) + \lambda \cdot u_i.$$

The planner is optimized via REINFORCE using the batch mean as a baseline to reduce variance:

$$\nabla_\theta \mathcal{J} \approx \frac{1}{N} \sum_{i=1}^{N} (R_i - b) \sum_{t=1}^{T} \nabla_\theta \log \tilde{\pi}_\theta(a_{t,i} \mid s_{t,i}),$$

where $b = \frac{1}{N} \sum_{j=1}^{N} R_j$ is the average reward of all trajectories in the current batch.

3.4 Executor: Frozen Action Execution

The executor is a separate, larger pretrained LLM (e.g., Qwen2.5-7B-Instruct) that remains strictly frozen. Its role is purely functional: given the prompt constructed by the planner $[s_t; \mathcal{T}(a_t)]$, it generates the reasoning content o_t. This separation ensures that improvements are attributable to the optimized *planning strategy* rather than parameter updates to the reasoning model itself.

4 Experiments

Our experiments aim to evaluate the effectiveness of the proposed PG-HAP framework and provide insights into the dynamics of stepwise high-level planning. Specifically, we address the following three research questions:

Q1: Does stepwise, policy-guided planning outperform both fixed-strategy prompting and instance-level planning baselines across diverse reasoning tasks?

Q2: How do the proposed training mechanisms (logit masking, diversity reward) and the specific design of the high-level action space contribute to the system's performance?

Q3: Does the learned planner exhibit interpretable, task-adaptive behaviors that mitigate the action collapse observed in prior methods?

4.1 Experimental Setup

Benchmarks and Evaluation. We evaluate our framework on five diverse reasoning benchmarks, categorized into mathematical reasoning (**MATH** (Hendrycks et al., 2021), **GSM8K** (Cobbe et al., 2021), **SVAMP** (Patel et al., 2021)) and commonsense reasoning (**CSQA** (Talmor et al., 2019), **StrategyQA** (Geva et al., 2021)). These datasets cover a wide spectrum of reasoning depths and structural complexities. We adhere to the standard training and test splits for all datasets; detailed statistics are provided in Appendix B. For evaluation, we employ the unified answer parser and accuracy metric provided by HiAR-ICL (Wu et al., 2024) across all methods to ensure a rigorous and fair comparison.

Baselines. We compare PG-HAP with representative baselines grouped into two categories. (1) **Prompting-based methods:** We reproduce Zero-shot CoT (Kojima et al., 2022) and Few-shot CoT (Wei et al., 2022) following their original papers, using the same frozen executor as our method. (2) **Action-based methods:** We reproduce CoM (Liu et al., 2025) following its original paper and evaluate HiAR-ICL[3] (Wu et al., 2024) and DOTS[4] (Yue et al., 2025) using their official implementations, with specific adaptations for fair comparison. For DOTS, we train the planner via full-parameter SFT on Qwen2.5-1.5B-Instruct with DOTS-generated data, matching the planner size of PG-HAP. For HiAR-ICL, we align the offline trajectory search model with the test-time executor. Crucially, while the original HiAR-ICL employs majority voting over the top-5 retrieved templates, we select only the top-1 action template to generate a single reasoning trajectory. This ensures a unified evaluation setting where all methods perform a single pass of reasoning. For comprehensive implementation details, please refer to Appendix C.

4.2 Overall Performance

We first evaluate whether learning a reinforcement-trained high-level action planner can consistently improve reasoning performance when the executor LLM is fully frozen (**Q1**). Table 2 reports the accuracy across five diverse benchmarks under two executor scales (Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct).

As shown in Table 2, PG-HAP delivers consistent performance gains, achieving the highest average accuracy across both executor scales.

Comparison with Prompting Baselines. PG-HAP consistently surpasses the best CoT baseline on datasets requiring structural flexibility or knowledge retrieval, achieving notable gains of 2.67% on SVAMP and 1.80% on CSQA with the 7B executor. While Zero-shot CoT retains an edge on MATH (78.84% versus 76.60%), likely due to the benefit of uninterrupted derivation chains, PG-HAP proves more effective on tasks like SVAMP and CSQA where problems benefit from intermediate verification or structured decomposition. This highlights a complementary trade-off: CoT excels at continuous generation, while PG-HAP excels at controlled and structured reasoning. We further show in Appendix G that this MATH gap closes with a stronger executor.

Comparison with Planning Baselines. Existing planning methods often struggle to compete with simple CoT—for instance, HiAR-ICL and DOTS achieve average accuracies of 76.27% and 75.77% on the 7B executor, both below Few-shot CoT's 83.79%. We identify two primary causes. First, instance-level methods like HiAR-ICL and DOTS suffer from inflexibility—by committing to a fixed template before reasoning begins, they cannot adapt if the template mismatches the problem, leading to notable drops on tasks requiring diverse reasoning strategies. Second, training-free CoM is hindered by unoptimized heuristics, where rigid prompting rules often induce structural redundancy. In contrast, PG-HAP employs learned stepwise decision-making, successfully bridging this gap to outperform all planning baselines.

Scalability with Stronger Executors. PG-HAP demonstrates strong scalability. When scaling from 3B to 7B, average accuracy improves by 7.14% (from 77.65% to 84.79%), comparable to CoT's gains. This indicates that as the frozen executor becomes more capable, our planner effectively leverages these enhanced capabilities. We further validate this trend with a next-generation executor (Qwen3-8B) in Appendix G, where PG-HAP continues to outperform CoT and notably surpasses it on MATH.

4.3 Ablation Study

To answer **Q2**, we investigate the individual contributions of the proposed training mechanisms and the specific design of the high-level action space.

4.3.1 Impact of Training Mechanisms

We first evaluate the two key regularization strategies: *Action-Dependency Masking (ADM)* and the *Action Diversity Reward (ADR)*. Table 3 summarizes the performance on GSM8K and StrategyQA using the Qwen2.5-7B executor.

[3] https://github.com/jinyangwu/HiARICL
[4] https://github.com/MurongYue/DOTS

Table 2: Accuracy (%) on reasoning benchmarks under different frozen executor models. **Bold** indicates the best result per column within each executor group. PG-HAP achieves the highest average accuracy across both executor scales, demonstrating consistent improvements.

Method	Executor Model	MATH	GSM8K	SVAMP	StrategyQA	CSQA	Average
Zero-shot CoT	Qwen2.5-3B-Instruct	**67.20**	86.35	88.67	64.19	76.65	76.61
Few-shot CoT		66.00	**86.50**	88.67	63.61	77.06	76.37
CoM		54.80	71.19	81.33	63.02	67.89	67.65
HiAR-ICL		62.40	72.18	84.33	64.19	73.96	71.41
DOTS		65.00	76.80	82.00	61.28	67.32	70.48
PG-HAP (Ours)		66.80	85.06	**90.33**	**68.27**	**77.81**	**77.65**
Zero-shot CoT	Qwen2.5-7B-Instruct	**78.84**	90.67	91.00	76.42	81.98	83.78
Few-shot CoT		78.70	91.28	92.00	76.13	80.83	83.79
CoM		70.60	87.87	89.67	71.47	77.64	79.45
HiAR-ICL		75.40	89.49	91.03	72.05	53.40	76.27
DOTS		72.80	89.10	87.30	50.21	79.44	75.77
PG-HAP (Ours)		76.60	**92.04**	**94.67**	**76.86**	**83.78**	**84.79**

Table 3: Ablation of training mechanisms. **ADM**: Action-Dependency Masking; **ADR**: Action Diversity Reward.

Setting	GSM8K	StrategyQA
PG-HAP (Full)	**92.04**	**76.86**
w/o ADM (Masking)	91.43	76.12
w/o ADR (Diversity)	89.46	75.90

The results highlight distinct roles for each mechanism: **ADM Enforces Valid Transitions:** Removing the logit mask (w/o ADM) leads to moderate but consistent performance drops (-0.61% on GSM8K, -0.74% on StrategyQA). While numerically smaller than the diversity reward's impact, this mechanism serves a foundational role: it prevents the planner from selecting logically impossible actions, such as attempting *Verification* before generating any reasoning steps. We further extend this ablation to the weaker Qwen2.5-3B executor, where accuracy drops are substantially larger (-3.03% on GSM8K vs. -0.61% with 7B), and qualitative collapse patterns are more severe. Full cross-executor results and behavioral analysis are provided in Appendix H.

ADR Prevents Action Collapse: The removal of the diversity reward (w/o ADR) causes the most significant degradation, particularly on GSM8K (-2.58%). Mathematical reasoning problems require diverse strategies, as simpler instances can often be solved via direct reasoning, whereas more complex problems benefit from structured decomposition. Without ADR, the planner tends to get stuck in a "shortcut" loop (e.g., always using direct reasoning) to maximize immediate rewards. The diversity reward explicitly penalizes this repetition, encouraging the planner to utilize the full set of available actions to solve harder instances.

4.3.2 Contribution of High-Level Actions

To understand how different cognitive capabilities contribute to reasoning, we perform a leave-one-out ablation study on the action space. We retrain the planner after removing one specific action category while keeping all other settings constant.

Table 4: Ablation of high-level actions on Qwen2.5-7B-Instruct. Performance drops indicate the relative importance of each cognitive capability.

Action Set	GSM8K	StrategyQA
Full Action Set	**92.04**	**76.86**
w/o Problem Analysis	90.14	73.94
w/o Problem Decomp.	91.05	76.27
w/o Knowledge Recall	90.67	75.40
w/o Code Gen./Ref.	91.05	–
w/o Verification	90.98	76.27

As shown in Table 4, the impact of each action varies by domain, revealing the task-dependent nature of planning:

Foundational Role of Analysis: *Problem Analysis* proves critical for both tasks, with its removal causing the largest drops on both StrategyQA (-2.92%) and GSM8K (-1.90%). This suggests that identifying key conditions and constraints before solving is a universal prerequisite for high-quality reasoning.

Domain-Specific Dependencies: *Knowledge Recall* is vital for StrategyQA (−1.46%), confirming that commonsense reasoning is more knowledge-intensive. Notably, it also impacts GSM8K (−1.37%), suggesting that explicit recall of mathematical properties or formulas helps stabilize the executor even in arithmetic tasks. Conversely, *Code Generation* contributes specifically to mathematical tasks (−0.99% on GSM8K). Note that this action is excluded from the StrategyQA experiments, as code generation is designed for computational tasks and is unnecessary for commonsense reasoning in StrategyQA.

Verification as a Safety Net: Removing *Verification* leads to consistent drops of approximately 1.0% on GSM8K and 0.6% on StrategyQA. Unlike *Analysis*, which is needed for almost every problem, *Verification* is a conditional action invoked only when the planner needs to correct errors or double-check intermediate results, serving as an effective safety net.

Synergy over Dominance: No single action removal causes a catastrophic failure, yet the full set yields the best performance. This indicates that the system's robustness stems from the *synergy* of diverse actions. The planner dynamically composes these capabilities—first analyzing the problem, retrieving relevant knowledge when necessary, and verifying calculation-heavy steps—enabling it to solve complex problems that no single strategy can address alone.

4.3.3 Isolating Learned vs. Enforced Components

A key question is whether PG-HAP's improvements stem from the RL-trained planner or merely from handcrafted structural constraints (ADM). To isolate these contributions, we compare PG-HAP against several non-RL planners operating under the same action space and transition mask (Table 5).

Table 5: Comparison of planner variants under identical ADM constraints (Qwen2.5-7B executor).

Planner Variant	GSM8K	StrategyQA
Random + ADM	84.51	72.10
Untrained + ADM	86.05	73.22
Best Fixed + ADM	89.16	75.40
DOTS (SFT)	89.10	50.21
PG-HAP (Ours)	**92.04**	**76.86**

Several findings emerge. First, the mask alone is insufficient: Random + ADM remains far below PG-HAP (−7.53% on GSM8K), confirming that valid transitions do not substitute for learned strategy selection. Second, Best Fixed + ADM underperforms PG-HAP by 2.88% on GSM8K, demonstrating the value of question-dependent adaptation. Third, DOTS performs competitively on GSM8K but collapses on StrategyQA (50.21%), whereas PG-HAP maintains robust performance across both domains. These results confirm that ADM defines which action sequences are *valid*, while the RL planner learns which valid sequence to *choose* for each question.

4.3.4 Cross-Domain Generalization

To assess whether the learned planning policy generalizes beyond its training distribution, we train the planner on CSQA (commonsense QA) and directly evaluate it on GPQA Diamond (Rein et al., 2023)—a graduate-level scientific reasoning benchmark—without any retraining or action space modification. The CSQA-trained planner improves over all CoT baselines by +3.04% (Appendix I), suggesting that PG-HAP learns general action-selection skills that transfer across domains rather than overfitting to dataset-specific patterns.

4.4 Planner Behavior Analysis

To address **Q3**, we analyze how the learned planner adapts its strategy to different reasoning tasks. Figure 3 visualizes the distribution of top action sequences, while the detailed statistics of individual actions are provided in Appendix A (Figure 4).

Task-Adaptive Planning Patterns. The comparison reveals clear differences in how PG-HAP approaches different domains, confirming its ability to learn context-aware strategies.

Mathematical Reasoning (GSM8K): Preference for Structure. As shown in Figure 3(a), the dominant action sequence is $\mathbf{A} \rightarrow \mathbf{CR} \rightarrow \mathbf{F}$ (**59.1%**). This preference is reflected in the individual action distribution (refer to Figure 4(a) in Appendix), where **A** (Analysis) accounts for **24.0%** of all steps. This suggests the planner prioritizes explicit structural analysis to decompose arithmetic problems, followed by **CR** (Continue Reasoning) to execute the solution based on the analyzed plan.

Commonsense Reasoning (StrategyQA): Preference for Efficiency and Retrieval. In contrast, StrategyQA (Figure 3(b)) favors efficiency. The

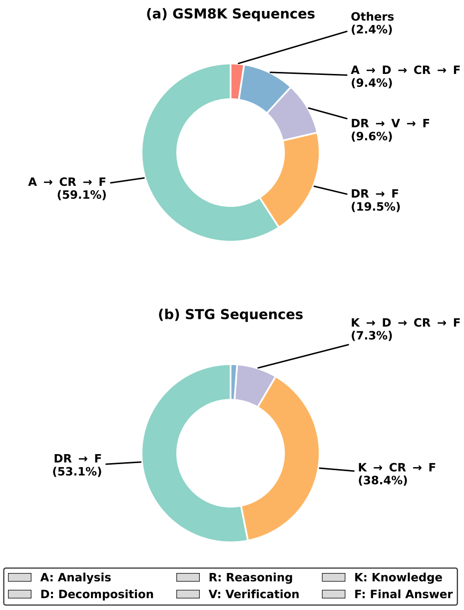

Figure 3: **Distribution of top reasoning action sequences on (a) GSM8K and (b) StrategyQA.** The planner exhibits distinct, task-adaptive strategies: favoring structured analysis paths (e.g., **A** $\rightarrow$ **CR**) for mathematics, while shifting to knowledge-centric paths (e.g., **K** $\rightarrow$ **CR**) for commonsense tasks. This pattern is further corroborated by the individual action distribution shown in Figure 4.

most frequent strategy is the efficient **DR** $\rightarrow$ **F** (**53.1%**), indicating that for over half the questions, the planner elects to reason directly without overhead. This is followed by **K** $\rightarrow$ **CR** $\rightarrow$ **F** (**38.4%**), where the planner first retrieves external facts (**K**) and then processes them via Continue Reasoning (**CR**). This aligns with the task nature and is supported by the high frequency of Knowledge Recall actions in the individual distribution (Figure 4(b) in Appendix), confirming that retrieving external knowledge is often more critical than structural decomposition in this domain.

Mitigation of Action Collapse. The visualizations confirm that PG-HAP avoids *dataset-level action collapse*. While dominant modes exist, the planner retains flexibility. For instance, on GSM8K, **9.6%** of trajectories follow the **DR** $\rightarrow$ **V** $\rightarrow$ **F** pattern. This indicates a dynamic correction behavior: the planner attempts to solve directly, detects uncertainty, and invokes **V** (Verification) before concluding. Such *stepwise adaptability*, including the ability to decide when to analyze, retrieve knowledge, or verify intermediate results, plays a central role in the robustness of PG-HAP.

MATH Failure Analysis. We analyze the 117 failure cases of PG-HAP (7B executor) on MATH. All failures reach the Final Answer action—none are caused by trajectory truncation. Among these, 65.2% produce incorrect intermediate reasoning, suggesting that stepwise segmentation can disrupt derivation continuity for long algebraic chains. Another 34.8% invoke Verification but still fail, as the errors were already present in earlier steps and Verification could not retroactively correct them. Along a separate dimension, 76.1% of failures concentrate on Level 4–5 problems requiring extended multi-step algebraic derivations. However, PG-HAP uniquely solves 43 problems that CoT fails, particularly on Intermediate Algebra where structured decomposition is beneficial. Consistent with the pattern noted in Section 4.2, CoT and PG-HAP show complementary strengths: CoT for uninterrupted symbolic derivation, PG-HAP for structured decomposition and verification. This gap closes with a stronger executor (Appendix G), confirming that the limitation is partly attributable to executor capability.

5 Conclusion

We proposed PG-HAP to address structural redundancy and action collapse in existing planning paradigms. By regulating a lightweight planner via *Action-Dependency Masking* and an *Action Diversity Reward*, our framework generates valid and diverse trajectories while keeping the executor frozen. PG-HAP consistently outperforms strong baselines across mathematical and commonsense benchmarks under multiple executor scales, and cross-domain transfer experiments show that the learned policy generalizes to unseen tasks without retraining. Behavioral analysis confirms that the planner develops interpretable, task-adaptive strategies. Future directions include automatic action discovery and lightweight co-adaptation between the planner and executor.

Limitations

While PG-HAP demonstrates strong performance, we identify two primary limitations that point toward future research directions:

Dependence on Frozen Executor Capabilities. Our framework focuses exclusively on optimizing the *planning policy*, relying on a frozen LLM for execution. Consequently, the upper bound of performance is limited by the executor's inherent ability to follow instructions and perform specific actions (e.g., precise calculation or code generation). As shown in our MATH failure analysis, stepwise segmentation can disrupt derivation continuity when the executor cannot maintain coherence across action boundaries. Future work could explore *lightweight co-adaptation*, such as tuning soft prompts or LoRA adapters for the executor to better align with the planner's action definitions.

Predefined and Coarse-Grained Action Space. The current action space consists of a fixed set of high-level cognitive operations derived from human priors. While our cross-domain transfer results (CSQA $\rightarrow$ GPQA) suggest these abstractions are reusable across domains, the coarse granularity may not capture nuanced, domain-specific strategies required for highly specialized tasks (e.g., theorem proving or legal reasoning). A promising direction is to automate *action discovery*, allowing the planner to expand or refine its action space dynamically during training.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (62472461), the Guangdong Basic and Applied Basic Research Foundation (2025A1515010129), the NTU Start-Up Grant (#023284-00001), Singapore, and the MOE AcRF Tier 1 Seed Funding Grant (#025041-00001, RS37/24).

References

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023.

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

Zhihao Dou, Yueming Feng, Jia Liu, and Zhigang Tan. 2025. Plan then action: High-level planning guidance reinforcement learning for llm reasoning. *arXiv preprint arXiv:2510.01833*.

Brown Ebouky, Andrea Bartezzaghi, and Mattia Rigotti. 2025. Eliciting reasoning in language models with cognitive tools. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *International Conference on Machine Learning, ICML 2023*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

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*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*.

Cong Liu, Jie Wu, Weigang Wu, Xu Chen, Liang Lin, and Wei-Shi Zheng. 2025. Chain of methodologies: Scaling test time computation without training. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 5298–5312, Vienna, Austria. Association for Computational Linguistics.

Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. 2025. S^2R: Teaching LLMs to self-verify and self-correct via reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22632–22654, Vienna, Austria. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Zhenting Qi, Mingyuan MA, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2025. Mutual reasoning makes smaller LLMs stronger problem-solver. In *The Thirteenth International Conference on Learning Representations*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158.

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 *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Jinyang Wu, Mingkuan Feng, Shuai Zhang, Feihu Che, Zengqi Wen, and Jianhua Tao. 2024. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *arXiv preprint arXiv:2411.18478*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, and 17 others. 2025. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Murong Yue, Wenlin Yao, Haitao Mi, Dian Yu, Ziyu Yao, and Dong Yu. 2025. DOTS: Learning to reason dynamically in LLMs via optimal reasoning trajectories search. In *The Thirteenth International Conference on Learning Representations*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Additional Statistics on Planner Behavior Analysis

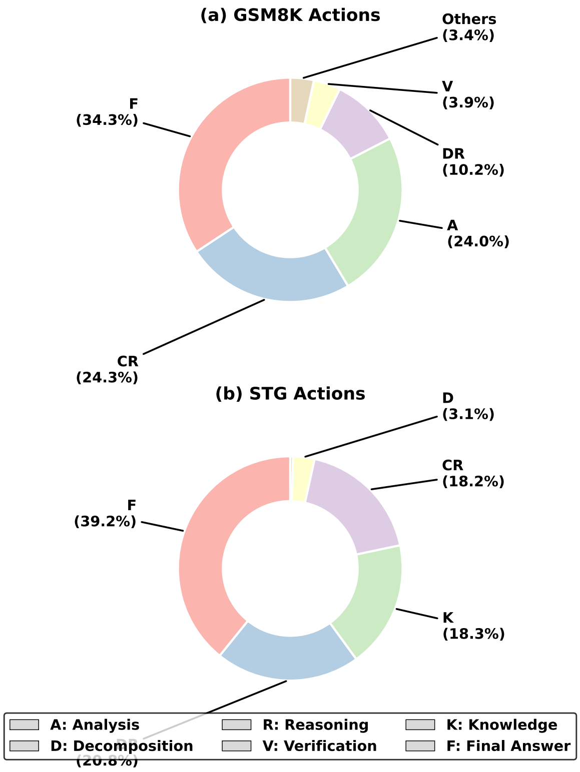

Figure 4: **Distribution of individual actions on (a) GSM8K and (b) StrategyQA.** Consistent with the sequence analysis in Figure 3, GSM8K is dominated by reasoning actions (**A**, **CR**, **DR**), whereas StrategyQA shows a significantly higher proportion of Knowledge Retrieval (**K**).

B Dataset Statistics and Splits

We evaluate our method on five diverse reasoning benchmarks, covering mathematical reasoning (GSM8K, MATH, SVAMP) and commonsense reasoning (StrategyQA, CSQA). For the training and validation sets, we adopt a standardized sampling strategy to ensure efficiency and consistency. Specifically, for large datasets, we randomly sample 2,000 instances for training and 500 for validation. For smaller datasets (e.g., SVAMP, StrategyQA), we reserve 200 instances for validation and use the remaining data for training. The detailed statistics for the training, validation, and testing splits used in our experiments are summarized in Table 6.

Table 6: Statistics of the datasets used in experiments. The training and validation sets are sampled from the original training splits, while the test sets remain consistent with the official benchmarks.

Dataset	**Train Set**	**Val Set**	**Test Set**
Mathematical Reasoning			
GSM8K	2,000	500	1,319
MATH	2,000	500	500
SVAMP	500	200	300
Commonsense Reasoning			
StrategyQA	1,403	200	687
CSQA	2,000	500	1,221

C Implementation Details

Model Configuration: We employ the Qwen2.5 family and Qwen3-8B for all experiments. Specifically, **Qwen2.5-3B-Instruct**, **Qwen2.5-7B-Instruct**, and **Qwen3-8B** (no-think mode) serve as the frozen executors. The planner is initialized from **Qwen2.5-1.5B-Instruct** (backbone frozen) with a trainable classification head predicting actions from the predefined set (e.g., *Analysis, Decomposition, Verification*). **Training:** The planner is trained on the training set of each benchmark using the REINFORCE algorithm. For each question, we sample 8 trajectories to estimate the expected reward. Optimization is performed for 2 epochs with a learning rate of 2×10^{-5} and a batch size of 2. We utilize action-dependency logit masking and an action diversity reward ($\lambda = 0.1$) to stabilize training. **Inference & Hardware:** During inference, we set the temperature to 0 for all methods to ensure deterministic reproducibility. Regarding the planning horizon, we set the maximum number of reasoning steps to 6 for CoM and 4 for PG-HAP, highlighting the efficiency of our approach. All experiments are conducted on two NVIDIA A100 (40GB) GPUs.

HiAR-ICL Reproduction Note: We identified a bug in the official HiAR-ICL codebase where the stop tokens for the OpenAI-compatible API were hardcoded as `["\n", "Answer"]` instead of using the caller-specified tokens. This caused premature truncation of multi-line reasoning outputs during MCTS search, disproportionately affecting tasks requiring extended reasoning chains (e.g., MATH). We fixed this by passing the correct stop tokens. Additionally, for CSQA we use the same first-letter answer normalization as all other methods to ensure a fair comparison, since HiAR-ICL's default fuzzy matching fails on answers in "a. text" format. All HiAR-ICL results reported in this paper use the corrected implementation.

D Algorithm Pseudocode

The training procedure of PG-HAP is summarized in Algorithm 5 below.

E Sample Reasoning Trajectory

The case study in Figure 6 and Figure 7 demonstrates complete inference trajectories generated by PG-HAP on GSM8K and StrategyQA test instances, respectively. The planner dynamically selects high-level actions, and the executor generates the corresponding reasoning steps.

F Statistical Significance Analysis

To verify the reliability of our improvements, we conduct paired bootstrap significance tests (B=10,000, seed=42) on per-sample predictions. Table 7 reports 95% confidence intervals and paired p-values for the Qwen2.5-7B executor.

Table 7: Bootstrap significance results (Qwen2.5-7B executor). †: $p < 0.05$; ‡: $p < 0.01$. –: bootstrap test not conducted due to a data mismatch identified post-hoc.

Dataset	Method	Acc.	95% CI	p vs. PG-HAP
GSM8K	PG-HAP	92.04	[90.52, 93.48]	–
	Zero-shot CoT	90.67	[89.08, 92.27]	0.029†
	Few-shot CoT	91.28	[89.69, 92.72]	0.156
SVAMP	PG-HAP	94.67	[92.00, 97.00]	–
	Zero-shot CoT	91.00	[87.67, 94.33]	0.009‡
	Few-shot CoT	92.00	[89.00, 95.00]	0.050
CSQA	PG-HAP	83.78	[81.74, 85.91]	–
	Zero-shot CoT	81.98	[79.85, 84.11]	0.035†
	Few-shot CoT	80.83	[78.62, 83.05]	0.001‡
StrategyQA	PG-HAP	76.86	[73.65, 80.06]	–
	Zero-shot CoT	76.42	[73.22, 79.62]	0.402
	Few-shot CoT	76.13	–	–

PG-HAP achieves statistically significant improvements ($p < 0.05$) over at least one CoT baseline on GSM8K, SVAMP, and CSQA. On StrategyQA, the absolute gain over Zero-shot CoT is small (+0.44%) and not significant, consistent with the observation that many StrategyQA questions are solvable via direct reasoning. MATH is excluded because its diverse answer formats (fractions, symbolic expressions, etc.) lead to inconsistent per-sample exact-match results across evaluation pipelines, making paired bootstrap testing unreliable.

We observe a similar pattern with the Qwen2.5-3B executor: PG-HAP achieves significant gains on StrategyQA ($p < 0.01$) where the weaker executor benefits most from structured planning, while differences on other benchmarks are not significant at the 3B scale.

G Evaluation with Qwen3-8B Executor

To further validate PG-HAP's scalability with next-generation executors, we evaluate with Qwen3-8B (no-think mode) on four benchmarks (Table 8).

Table 8: Accuracy (%) with Qwen3-8B executor. PG-HAP outperforms CoT on all four benchmarks and notably surpasses it on MATH.

Method	MATH	SVAMP	StrategyQA	CSQA
Zero-shot CoT	82.40	93.00	69.14	82.43
Few-shot CoT	81.60	93.00	69.87	81.49
PG-HAP (Ours)	**82.80**	**94.33**	**73.51**	**83.46**

PG-HAP continues to outperform CoT across all four benchmarks, with particularly notable gains on StrategyQA (+4.37% over Zero-shot CoT). Importantly, PG-HAP now surpasses CoT on MATH (82.80% vs. 82.40%), closing the gap observed with Qwen2.5 executors. This indicates that the planner's benefit persists and even strengthens as the executor becomes more capable, and that the MATH disadvantage is partly attributable to executor capability rather than a fundamental limitation of stepwise planning.

H Cross-Executor ADM Ablation

To further understand ADM's role across executor scales, we extend the ablation to the weaker Qwen2.5-3B executor (Table 9). The accuracy drops are substantially larger with 3B (-3.03% on GSM8K vs. -0.61% with 7B), revealing that ADM is especially critical when the executor cannot compensate for collapsed planning.

Table 9: Cross-executor ADM ablation. Accuracy drops are consistently larger with the weaker 3B executor, indicating that ADM is more critical when executor capability is limited.

Dataset	Executor	PG-HAP	No ADM	Δ
GSM8K	7B	92.04	91.43	−0.61
GSM8K	3B	85.06	82.03	−3.03
StrategyQA	7B	76.86	76.12	−0.74
StrategyQA	3B	68.27	67.25	−1.02

Qualitatively, collapse patterns differ across executor scales. Without ADM, the 7B executor collapses to minimal-length shortcuts: 98.79% of

GSM8K samples follow Direct Reasoning → Final Answer, while structured decomposition drops from 68.54% to 0.30%. The 3B executor exhibits a more severe failure: 96.3% of GSM8K samples contain self-loops (e.g., Continue Reasoning → Continue Reasoning), and only 3 of 7 actions are ever used. This demonstrates that ADM prevents reward-driven planning collapse rather than dictating a specific reasoning strategy.

I Cross-Domain Generalization Details

We train the planner on CSQA (commonsense QA) and directly evaluate it on GPQA Diamond (Rein et al., 2023)—a graduate-level scientific reasoning benchmark substantially different from commonsense QA—without any retraining or action space modification.

Table 10: Zero-shot cross-domain transfer: CSQA-trained planner evaluated on GPQA Diamond (Qwen2.5-7B executor). PG-HAP improves over CoT baselines despite the significant domain shift.

Method	**GPQA Diamond**
Zero-shot CoT	31.30
Few-shot CoT	30.80
PG-HAP (CSQA-trained)	**34.34**

Despite the difficulty shift, the CSQA-trained planner improves over all CoT baselines by +3.04% (Table 10). This suggests that PG-HAP learns general action-selection skills—such as when to analyze, retrieve knowledge, or verify—that transfer across domains, rather than overfitting to dataset-specific patterns. The same manually defined action space and transition mask are reused without modification, indicating that these abstractions function as reusable, domain-independent reasoning primitives.

J Prompt Templates

Algorithm 1: PG-HAP Training Procedure

Input: Frozen Executor $\mathcal{E}$, Planner Policy π_θ, Training Dataset $\mathcal{D}$
Hyperparameters: Batch size N, Max steps $T_{\max}$, Diversity weight λ

---

1. **Initialize:** Training loop
2. **While** not converged **do**
   (a) Sample a batch of questions $Q = \{q_1, \ldots, q_N\}$ from $\mathcal{D}$
   (b) Initialize batch trajectories $\mathcal{B} \leftarrow \emptyset$
   (c) *// Step 1: Parallel Rollout*
   (d) **For** $i = 1$ to N **do**
      i. Initialize state $s_0 \leftarrow q_i$, trajectory $\tau_i \leftarrow \emptyset$
      ii. **For** $t = 1$ to $T_{\max}$ **do**
         A. Get hidden states from Planner backbone given s_{t-1}
         B. Compute action logits via the trainable head
         C. Apply **Action-Dependency Mask** $\mathbf{M}_t$ (See Table 1)
         D. Sample action $a_t \sim \text{Softmax}(\text{logits} + \mathbf{M}_t)$
         E. **If** a_t is *Final Answer* or $t = T_{\max}$ **then**
            - Generate final answer and terminate
            - $\tau_i \leftarrow \tau_i \cup \{(s_{t-1}, a_t)\}$
            - **Break** loop
         F. **End If**
         G. Construct prompt template $\mathcal{P} = [s_{t-1}; \mathcal{T}(a_t)]$
         H. Generate reasoning content $o_t \leftarrow \mathcal{E}(\mathcal{P})$ *(Frozen Executor)*
         I. Update state $s_t \leftarrow [s_{t-1}, a_t, o_t]$
         J. $\tau_i \leftarrow \tau_i \cup \{(s_{t-1}, a_t)\}$
      iii. **End For**
      iv. Add τ_i to $\mathcal{B}$
   (e) **End For**
   (f) *// Step 2: Reward Calculation*
   (g) **For** $i = 1$ to N **do**
      i. Calculate task correctness reward $R_{\text{task}} \in \{0, 1\}$
      ii. Calculate **Diversity Reward** u_i based on sequence uniqueness in $\mathcal{B}$
      iii. Total Reward $R_i \leftarrow R_{\text{task}} + \lambda \cdot u_i$
   (h) **End For**
   (i) *// Step 3: Policy Optimization*
   (j) Compute baseline $b \leftarrow \frac{1}{N} \sum_{j=1}^{N} R_j$
   (k) Compute loss $\mathcal{L} \leftarrow -\frac{1}{N} \sum_{i=1}^{N} (R_i - b) \sum_{(s,a) \in \tau_i} \log \pi_\theta(a \mid s)$
   (l) Update Planner parameters θ using Gradient Descent
3. **End While**

Figure 5: PG-HAP training procedure.

Case Study: GSM8K Reasoning Process

Question: Jill gets paid \$20 per hour to teach and \$30 to be a cheerleading coach. If she works 50 weeks a year, 35 hours a week as a teacher and 15 hours a week as a coach, what's her annual salary?

▶ Action: Problem Analysis

> **Given:** Hourly pay: \$20 (teaching), \$30 (coaching). Works 50 weeks/year, 35 hrs/week teaching, 15 hrs/week coaching. **Goal:** Determine Jill's total annual salary. **Constraints:** Salary = hourly rate $\times$ weekly hours $\times$ weeks per year.

▶ Action: Problem Decomposition

> Planned steps: (1) Calculate weekly teaching earnings. (2) Calculate weekly coaching earnings. (3) Sum weekly earnings. (4) Multiply by 50 weeks for annual salary.

▶ Action: Continue Reasoning

> **Step 1:** Weekly teaching earnings: $20 \times 35 = 700$ dollars/week. **Step 2:** Weekly coaching earnings: $30 \times 15 = 450$ dollars/week. **Step 3:** Total weekly earnings: $700 + 450 = 1150$ dollars/week. **Step 4:** Annual salary: $1150 \times 50 = 57500$ dollars/year. $\boxed{57500}$

▶ Action: Final Answer Conclusion

> $\boxed{57500}$

Figure 6: GSM8K reasoning trajectory example.

Case Study: StrategyQA Reasoning Process

Question: Is B's place in alphabet same as Prince Harry's birth order?

▶ Action: Knowledge Recall

- **Alphabet:** The letter B is the second letter.
- **History:** Prince Harry is the second son of Charles, Prince of Wales.

▶ Action: Continue Reasoning

Step 1: Determine the position of the letter B in the alphabet. The letter B is the second letter.
Step 2: Determine Prince Harry's birth order. Prince Harry is the second son of Charles, Prince of Wales.
Step 3: Compare the two positions. Since both the letter B and Prince Harry's birth order are second, they are the same.
Final Answer: $\boxed{true}$

▶ Action: Final Answer Conclusion

$\boxed{true}$

Figure 7: StrategyQA reasoning trajectory example.

```
Problem Analysis

You are given a question and (if any) a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: Analyze the problem without solving it.

Requirements for Problem Analysis:
- Restate the problem in your own words.
- Identify all given inputs (Given).
- Identify what needs to be found or proven (Goal).
- List any constraints inherent to the problem (Constraints).
- Output your analysis in a single `<step>...</step>` block, preceded by "Your response:".

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Reasoning history:
(none)

Your response:
<step>
The problem involves calculating the total remaining chocolates after some are eaten.
Given:
- Leah had 32 chocolates.
- Her sister had 42 chocolates.
- Together they ate 35 chocolates.
Goal: Find the total number of chocolates left.
Constraints: Subtraction must reflect that total remaining cannot be negative.
</step>

---

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday.
      How many computers are now in the server room?
Reasoning history:
(none)

Your response:
<step>
The problem involves computing the final number of computers after repeated daily additions.
Given:
- Initial number of computers = 9.
- 5 computers installed per day.
- Installation lasted 4 days (Monday to Thursday).
Goal: Find the total number of computers in the server room after the installations.
Constraints: Installation is additive and happens every day for 4 days.
</step>
```

Figure 8: Prompt template for Problem Analysis.

Background Recall

```
You are given a question and, if available, a reasoning history consisting of previous steps enclosed in `<step>...</step>`
      tags.

Task: Recall **only relevant background knowledge, facts, formulas, or standard rules** that may help solve the problem.

Strict Requirements:
- **Do NOT perform any calculations, algebraic substitutions, variable definitions, or stepwise reasoning.** This is purely a
      recall step.
- Provide concise, general formulas, theorems, identities, or standard solution strategies.
- Keep output short: **15 bullet-like items**, each 1 sentence or a short formula.
- If multiple domains are relevant (e.g., geometry + algebra), label each bullet by domain.
- Output MUST be a single `<step>...</step>` block, preceded exactly by the line: `Your response:`.

---

### Example
Question: A rectangle has a length that is 3 times its width. If the perimeter is 48 units, what are the dimensions of the
      rectangle?
Reasoning history: (none)

Your response:
<step>
- Geometry: Rectangle perimeter formula: \(P = 2(\text{length} + \text{width})\).
- Algebra: Express one variable in terms of another if related; solve resulting linear equation.
- Strategy: Solve linear equations to find unknown dimensions.
</step>

---

Question: A theater sells 120 tickets on the first day. Each subsequent day, 10 fewer tickets are sold than the previous day.
      How many tickets are sold in total over 5 days?
Reasoning history: (none)

Your response:
<step>
- Sequences: Arithmetic sequence nth term: \(a_n = a_1 + (n-1)d\).
- Sums: Sum of first n terms: \(S_n = \frac{n}{2}(a_1 + a_n)\) or \(S_n = n a_1 + \frac{n(n-1)}{2} d\).
- Strategy: Use formulas to compute totals over fixed steps.
</step>

---

Question: A car accelerates uniformly from 0 to 20 m/s in 5 seconds. What is its acceleration and distance traveled?
Reasoning history: (none)

Your response:
<step>
- Physics: Uniform acceleration formulas: \(v = v_0 + at\), \(s = v_0 t + \frac{1}{2} a t^2\).
- Strategy: Use the above kinematic equations to find acceleration and displacement.
- Units: Keep consistent SI units for velocity (m/s), time (s), and distance (m) when applying formulas.
</step>
```

Figure 9: Prompt template for Background Recall.

Problem Decomposition

```
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: Break the problem into smaller logical steps without solving them.

Requirements for Problem Decomposition:
- Divide the problem into a clear sequence of sub-steps.
- Ensure each sub-step is logically coherent.
- Do not perform any calculations; this is a planning step.
- Output your plan in a single `<step>...</step>` block, preceded by "Your response:".

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Reasoning history:
<step>
The problem involves calculating the total remaining chocolates after some are eaten.
Given:
- Leah had 32 chocolates.
- Her sister had 42 chocolates.
- Together they ate 35 chocolates.
Goal: Find the total number of chocolates left.
Constraints: Subtraction must reflect that total remaining cannot be negative.
</step>

Your response:
<step>
Planned reasoning steps:
1. Add Leah's chocolates and her sister's chocolates to find the initial total.
2. Subtract the number of chocolates eaten from this total.
3. Report the number of chocolates remaining.
</step>

---

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday.
      How many computers are now in the server room?
Reasoning history:
(none)

Your response:
<step>
Planned reasoning steps:
1. Calculate how many computers were installed in total across 4 days.
2. Add this number to the initial 9 computers.
3. Report the final total number of computers.
</step>
```

Figure 10: Prompt template for Problem Decomposition.

Continue Reasoning

```
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: Carry out the remaining reasoning steps, continuing from the previous reasoning history.
- Follow logically from the given steps until the final answer.
- Output must begin with "Your response:", use a single `<step>...</step>` block, and end with the final answer in
      `\boxed{}`.

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Reasoning history:
<step>
The problem involves calculating the total remaining chocolates after some are eaten.
Given:
- Leah had 32 chocolates.
- Her sister had 42 chocolates.
- Together they ate 35 chocolates.
Goal: Find the total number of chocolates left.
Constraints: Subtraction must reflect that total remaining cannot be negative.
</step>

Your response:
<step>
Let's continue reasoning step by step based on the problem analysis.
Step 1: Total chocolates initially = 32 + 42 = 74.
Step 2: Subtract the chocolates eaten: 74 - 35 = 39.
Final Answer: \boxed{39}
</step>

---

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday.
      How many computers are now in the server room?
Reasoning history:
<step>
The problem involves computing the final number of computers after repeated daily additions.
Given:
- Initial number of computers = 9.
- 5 computers installed per day.
- Installation lasted 4 days (Monday to Thursday).
Goal: Find the total number of computers in the server room after the installations.
Constraints: Installation is additive and happens every day for 4 days.
</step>
<step>
Planned reasoning steps:
1. Calculate how many computers were installed in total across 4 days.
2. Add this number to the initial 9 computers.
3. Report the final total number of computers.
</step>

Your response:
<step>
Let's execute the planned steps.
Step 1: Total installed computers = 5 * 4 = 20.
Step 2: Add this to the initial 9 computers: 9 + 20 = 29.
Step 3: Final Answer: \boxed{29}
</step>
```

Figure 11: Prompt template for Continue Reasoning.

Direct Reasoning

```
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: Start reasoning directly from the question (history will always be empty in this case).
- Begin your reasoning with "Let's think step by step."
- Provide a clear chain of thought until the final answer.
- Output must begin with "Your response:", use a single `<step>...</step>` block, and end with the final answer in
      `\boxed{}`.

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Reasoning history:
(none)

Your response:
<step>
Let's think step by step.
Leah had 32 chocolates, her sister had 42, so together they had 32 + 42 = 74.
They ate 35 chocolates, so 74 - 35 = 39.
Final Answer: \boxed{39}
</step>

---

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday.
      How many computers are now in the server room?
Reasoning history:
(none)

Your response:
<step>
Let's think step by step.
Initially there were 9 computers.
Each day, 5 more were installed, for 4 days: 5 * 4 = 20.
Adding this to the original 9: 9 + 20 = 29.
Final Answer: \boxed{29}
</step>
```

Figure 12: Prompt template for Direct Reasoning.

Code Generation

````
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: Write Python code to perform one or more reasoning steps, especially for computation or symbolic manipulation.

Requirements for Code Generation:
- Enclose all code in one single ```python ... ``` block.
- Ensure syntax is correct.
- MUST end with a print() statement to display the execution result.
- Output the code in a single `<step>...</step>` block, preceded by "Your response:".
- Before executing the program, you have no idea of the final answer. Don't show it in your comment or code. And don't use
      the plot function.

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Reasoning history:
(none)

Your response:
<step>
```python
leah = 32
sister = 42
eaten = 35
total_remaining = (leah + sister) - eaten
print("Total remaining chocolates:", total_remaining)

```

</step>

---

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday.
      How many computers are now in the server room?
Reasoning history:
<step>
Given:

* Initial computers = 9
* 5 computers installed per day
* Installation lasted 4 days
Goal: Find total number of computers after installations

</step>

Your response:
<step>

```python
initial_computers = 9
daily_install = 5
days = 4
total_computers = initial_computers + daily_install * days
print("Total computers in the server room:", total_computers)

```

</step>
````

Figure 13: Prompt template for Code Generation.

Code Refinement

````
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags. If a history step contains Python code, it will be followed by `Execution Results: { ... }`
      showing the output or error.

Your task: Revise previously generated code that produced incorrect results or errors, explain the issue, and provide a
      corrected version.

Requirements for Code Refinement:

* Identify and explain the error in the original code based on the Execution Results.
* Provide a corrected Python code snippet enclosed in a single `python ... ` block.
* Ensure the corrected code ends with a print() statement displaying the execution result.
* Output the explanation and corrected code in a single `<step>...</step>` block, preceded by "Your response:".

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

Reasoning history:
<step>

```python
total_remaining = leah + sister - eaten
print("Total remaining chocolates:", total_remaining)

```

</step>
Execution Results: NameError: name 'leah' is not defined

Your response: <step>
The previous code caused a NameError because the variables `leah`, `sister`, and `eaten` were not defined before usage.
      Corrected code:

```python
leah = 32
sister = 42
eaten = 35
total_remaining = (leah + sister) - eaten
print("Total remaining chocolates:", total_remaining)

```

</step>

---

{An example is omitted here due to space constraints.}

</step>
````

Figure 14: Prompt template for Code Refinement.

Verification

```
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: Review the previous reasoning steps, identify any errors, and provide corrections if necessary.

Requirements for Self-Evaluation:

* Perform a step-by-step evaluation of the reasoning.
* For each step, indicate if it is correct or identify the error.
* Only provide corrections for steps that are wrong; do not repeat correct steps in detail.
* Give the final corrected answer after applying necessary fixes.
* Output everything in a single `<step>...</step>` block, preceded by "Your response:".

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

Reasoning history:
<step>
Total chocolates initially = 32 + 42.

Subtract chocolates eaten: 74 - 36. Final answer = 38.
</step>

Your response:
<step>
Step-by-step evaluation:

* Step 1: Correct, total chocolates = 74.
* Step 2: Incorrect, subtracted 36 instead of 35.
* Step 3: Result is incorrect due to Step 2.

Corrected reasoning:

* Correct Step 2: 74 - 35 = 39. Final answer: \boxed{39}.

</step>

---

Question: Circle T has its center at point T(-2,6). Circle T is reflected across the y-axis and then translated 8 units down.
      What are the coordinates of the image of the center of circle T?

Reasoning history:
<step>
Reflect across y-axis: (-(-2),6) = (2,6).

Translate down 8 units: (2,6+8) = (2,14).

Final coordinates: (2,14)
</step>

Your response:
<step>
Step-by-step evaluation:

* Reflection across y-axis: Correct.
* Translation down 8 units: Incorrect; should subtract 8 from y-coordinate.
* Final coordinates: Incorrect due to Step 2.

Corrected reasoning:

* Keep reflection as is.
* Correct translation: (2,6-8) = (2,-2).
Final coordinates: \boxed{(2,-2)}.

</step>

---

{An example is omitted here due to space constraints.}

</step>
```

Figure 15: Prompt template for Verification.

Final Answer Conclusion

```
You are given a question and a reasoning history consisting of previous reasoning steps. Each step is enclosed in
      `<step>...</step>` tags.

Your task: State the final answer based on the preceding reasoning.

Requirements for Final Answer Conclusion:

* Output only the final answer in concise boxed format: `\boxed{...}`.
* Do not include any additional reasoning or steps.
* Output in a single `<step>...</step>` block, preceded by "Your response:".

---

### Example

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

Reasoning history:
<step>
Total chocolates initially = 32 + 42 = 74.

Subtract chocolates eaten: 74 - 35 = 39.
</step>

Your response:
<step>
\boxed{39}
</step>

---

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday.
      How many computers are now in the server room?

Reasoning history:
<step>
Initial computers = 9.

Total installed over 4 days = 5 * 4 = 20.

Total computers = 9 + 20 = 29.
</step>

Your response:
<step>
\boxed{29}
</step>
```

Figure 16: Prompt template for Final Answer Conclusion.