Learn Like Humans: Use Meta-cognitive Reflection for Efficient Self-Improvement

Anonymous ACL submission

Abstract

While Large Language Models (LLMs) enable complex autonomous behavior, current agents remain constrained by static, human-designed prompts that limit adaptability. Existing self-improving frameworks attempt to bridge this gap but typically rely on inefficient, multi-turn recursive loops that incur high computational costs. To address this, we propose **M**etacognitive **A**gent **R**eflective **S**elf-improvement (MARS), a framework that achieves efficient self-evolution within a single recurrence cycle. Inspired by educational psychology, MARS mimics human learning by integrating principle-based reflection (abstracting normative rules to avoid errors) and procedural reflection (deriving step-by-step strategies for success). By synthesizing these insights into optimized instructions, MARS allows agents to systematically refine their reasoning logic without continuous online feedback. Extensive experiments on six benchmarks demonstrate that MARS outperforms state-of-the-art self-evolving systems while significantly reducing computational overhead. Code are available at https://anonymous.4open.science/r/MARS-9F16

1 Introduction

Large language models (LLMs) have enabled autonomous agents capable of complex reasoning, planning, and tool use (Brown et al., 2020; Wei et al., 2022; Yao et al., 2023). However, current agents rely on fixed, human-designed components—manually crafted prompts, predefined workflows, and static configurations—limiting their adaptability to strategies within human intuition (Hu et al., 2025a; Wang et al., 2024). While machine learning history shows hand-designed solutions are consistently replaced by learned ones (Elsken et al., 2019; Zoph and Le, 2017), agent development remains largely manual. The theoretical basis for self-improving AI dates back to Schmidhuber's

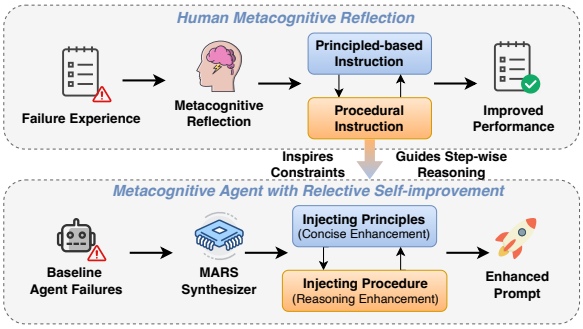


Figure 1: The Cognitive Inspiration behind MARS. This framework parallels human reflection with the MARS (Metacognitive Agent with Reflective Self-improvement) agent, converting baseline agent failures into principled-based and procedural instructions to synthesize enhanced prompts.

Gödel machines (Schmidhuber, 2007), which formalized self-referential systems that rewrite their own code. Although formal proof requirements make the original framework impractical (Steunebrink and Schmidhuber, 2011), it has inspired modern self-improving systems that rely on empirical validation instead.

However, current self-improvement frameworks for LLM agents tend to be constrained by multi-turn recursiveness, which results in inefficient learning and adaptation, as well as excessive computational resource usage. Humans, by contrast, are able to resolve previous errors and adapt to new solutions more efficiently through structured learning approaches. Research in education science has identified two complementary paradigms for guiding learners (Hiebert and Lefevre, 1986; Anderson, 1983). The first is *principle-based learning*, which focuses on helping learners avoid mistakes by establishing conceptual categories of what is correct versus incorrect, and understanding the underlying rules that govern a domain (Hiebert and Lefevre, 1986; Rittle-Johnson et al., 2001). The second is *procedural learning*, which emphasizes

using prior experience and step-by-step reasoning to increase the likelihood of successful outcomes (Anderson, 1983; Kolb, 1984). Rather than learning in isolation, humans benefit most when they integrate both approaches through systematic reflection and summarization of their experiences. Studies in metacognition have shown that structured reflection—where learners explicitly analyze what worked, what failed, and why—significantly improves learning efficiency and knowledge transfer (Flavell, 1979; Kaplan et al., 2013; Stanton et al., 2021). Furthermore, research on productive failure demonstrates that learning from one's own errors, when properly guided, leads to deeper conceptual understanding than direct instruction alone (Kapur, 2014, 2010).

In this work, we propose **MARS** (**M**etacognitive **A**gent with **R**eflective **S**elf-improvement), a framework that enables multi-agent systems to achieve efficient self-improvement within a single recurrence cycle by integrating both principle-based and procedural learning approaches. Inspired by human metacognitive learning, MARS allows agents to systematically reflect on their experiences, extracting general principles that help avoid past mistakes while simultaneously deriving procedural knowledge that replicates successful strategies. Unlike existing self-evolving agent frameworks that rely on multi-turn recursive improvement, which often leads to inefficient learning and excessive computational costs, MARS consolidates the learning process through structured summarization, enabling agents to maximize adaptation efficiency in each improvement cycle.

Our main contributions are as follows:

- We propose MARS, a self-improvement framework for multi-agent systems that integrates principle-based and procedural learning inspired by human meta-cognitive theory.
- We introduce a triple-pathway reflection mechanism that extracts: (1) normative principles for error avoidance, (2) procedural strategies for success replication, and (3) a unified synthesis of both pathways.
- We design a structured summarization module that consolidates learning within a single cycle, reducing computational overhead from multi-turn recursive improvement.
- We conduct extensive experiments on challenging knowledge and reasoning benchmarks, showing MARS outperforms existing self-evolving frameworks while requiring fewer iterations.

2 Related Work

Recent research has transitioned from static prompting to *self-evolving agents*—systems capable of analyzing their own performance, learning from errors, and modifying their behavior to improve over time. Drawing from meta-learning principles (Finn et al., 2017; Hospedales et al., 2022), these approaches can be broadly categorized into two paradigms: verbal reflection and structural self-modification.

The first category utilizes verbal reflection to facilitate learning from failure. *Reflexion* (Shinn et al., 2023) introduced this paradigm by enabling agents to generate natural language critiques of their mistakes, storing them in episodic memory to guide future reasoning. *RISE* (Qu et al., 2024) extends this by training models to iteratively detect and correct errors across multiple turns, demonstrating that self-correction capabilities can be internalized through fine-tuning. While effective, these methods primarily rely on inference-time recursion or memory retrieval rather than permanent parameter or prompt optimization. The second category focuses on automated architecture and code evolution. Systems like *ADAS* (Hu et al., 2025a) employ meta-agents to iteratively generate and evaluate new agent designs in code, while *AgentSquare* (Shang et al., 2025) adopts a modular approach to evolve components for planning, reasoning, and tool use. Similarly, *Agent-Pro* (Zhang et al., 2024) optimizes policies through reflection on historical trajectories. Taking this further, fully self-referential approaches allow agents to modify their own underlying source code. The *Gödel Agent* (Yin et al., 2025) enables agents to rewrite their logic guided by high-level objectives, a concept extended by the *Darwin Gödel Machine* (Zhang et al., 2025) and *SICA* (Robeyns et al., 2025), which integrate evolutionary search to explore diverse self-improvement paths.

Despite these advancements, current frameworks face significant efficiency bottlenecks. Reflection-based methods often depend on computationally expensive multi-turn recursive loops, while code-modifying agents require complex validation environments. Unlike these approaches, our framework draws inspiration from human metacognitive the-

ory to achieve efficient self-improvement within a *single recurrence cycle*.

3 Methodology

To achieve efficient self-improvement without the computational overhead of recursive loops, we propose MARS, a three-phase framework designed to systematically transform sporadic model failures into targeted, actionable prompt enhancements. Rather than treating errors as isolated incidents, our approach aggregates failures to identify systematic weaknesses, synthesizes remediation strategies, and integrates them into a self-improving loop. Figure 2 illustrates the complete pipeline.

The framework operates as follows. In the *Evaluation* phase, an analyzer model $\mathcal{M}_\phi$ examines each failed question and produces a structured analysis $\mathcal{A}_i$ capturing both question characteristics (type τ_i, topics $\mathcal{T}_i$) and failure attributes (error type ϵ_i, root cause ρ_i, specific mistake μ_i). The *Failure Allocation* phase then applies a grouping function κ to partition analyses into groups $\mathcal{G} = \{G_j\}$ based on shared type-topic keys, aggregating diagnostic attributes into group-level error profiles Ψ_j. Finally, the *Enhancement Generation* phase synthesizes targeted enhancements $(E_j^{(c)}, E_j^{(r)})$ for each group and combines them with the base prompt P via weighted aggregation to produce enhanced prompts $P'^{(c)}$ and $P'^{(r)}$. We detail each phase below.

3.1 Evaluation

The first phase of our enhancement pipeline performs fine-grained diagnosis of each incorrectly answered question. Rather than treating failures as a homogeneous set, we analyze each instance independently to understand the precise reasoning breakdown that led to the incorrect response.

Formally, let $\mathcal{Q} = \{q_i\}_{i=1}^n$ denote a set of failed questions from the benchmark evaluation. Each instance q_i comprises question text, options, ground-truth answer a_i^*, the model's predicted answer $\hat{a}_i$, and generated reasoning trace. We employ a specialized analyzer model, $\mathcal{M}_\phi$, to dissect these components. For every q_i, the analyzer produces a structured analysis

$$\mathcal{A}_i = (\tau_i, \mathcal{T}_i, \epsilon_i, \rho_i, \mu_i) \tag{1}$$

, which encapsulates two distinct categories of attributes:

The first category characterizes question itself. It assigns a "question type" $\tau_i \in \mathcal{Y}$ (where $\mathcal{Y} =$ {factual, conceptual, calculation, application}) and identifies a set of "topics" $\mathcal{T}_i \subseteq \mathcal{D}$ derived from domain vocabulary $\mathcal{D}$. As detailed in Table 1, the combination of question type τ_i and topic $\mathcal{T}_i$ serves as composite key for grouping failures in the subsequent Allocation phase. The second category characterizes specific error mechanism. This includes an "error type" $\epsilon_i \in \mathcal{E}$, a natural language "root cause" ρ_i explaining the fundamental reasoning deficit, and a "specific mistake" μ_i pinpointing the exact step where logic diverged. The error taxonomy $\mathcal{E}$, presented in Table 2, defines six categories ranging from conceptual misunderstandings to calculation errors.

To ensure consistent classification, we enforce a strict exclusivity rule: each failure is assigned to exactly one category in $\mathcal{E}$. In cases where multiple failure modes co-occur (e.g., a calculation error stemming from a conceptual misunderstanding), the analyzer assigns the category corresponding to the *earliest point of divergence* in the reasoning chain. The final output of this phase is the collection of structured analyses $\mathbb{A} = \{\mathcal{A}_i\}_{i=1}^n$.

3.2 Failure Allocation

This aggregation transforms sparse, per-instance observations into dense, group-level patterns. By clustering failures with shared characteristics, the allocation phase enables the subsequent system to generate high-level guidance that addresses classes of errors simultaneously, ensuring that the enhanced prompts are both targeted and scalable.

The second phase organizes individual error analyses into semantically coherent groups to enable pattern discovery. While the previous phase examined each failure in isolation, this phase identifies structural similarities across failures that may share common remediation strategies. We define a composite grouping function

$$\kappa : \mathbb{A} \to \mathcal{Y} \times 2^{\mathcal{D}}, \quad \kappa(\mathcal{A}_i) = (\tau_i, \mathcal{T}_i) \tag{2}$$

that maps each analysis to its "type-topic key" via $\kappa(\mathcal{A}_i) = (\tau_i, \mathcal{T}_i)$. This two-dimensional grouping captures the intuition that errors on calculation questions about thermodynamics likely stem from different causes than errors on conceptual questions about molecular biology, and thus require distinct enhancement strategies.

Given the analysis set $\mathbb{A}$ from the previous phase, we construct a partition

$$\Psi_j = (\mathcal{E}_j, \mathcal{R}_j, \mathcal{F}_j) \tag{3}$$

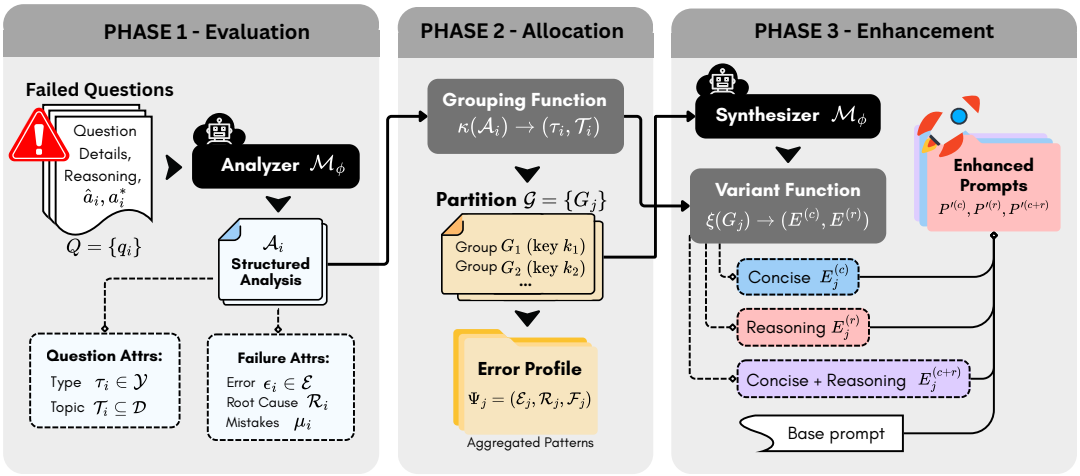


Figure 2: Overview of the proposed framework: (1) diagnose failed questions into structured analyses $\mathcal{A}_i$, (2) group by type-topic keys and aggregate error profiles Ψ_j, and (3) synthesize enhancements via weighted aggregation to produce $P'^{(c)}$ and $P'^{(r)}$.

Type	Description	Example
Factual	Recall of specific facts or definitions	"What is the atomic number of carbon?"
Conceptual	Understanding of principles or theories	"Why does entropy increase in isolated systems?"
Calculation	Quantitative problem requiring computation	"Calculate the final velocity given..."
Application	Applying knowledge to novel scenarios	"Which treatment would be most effective for..."
Analysis	Breaking down complex information	"What can be inferred from this experimental data?"
Comparison	Evaluating similarities or differences	"Which compound has higher boiling point?"

Table 1: Question type categories.

, where each group $G_j = \{\mathcal{A}_i \in \mathbb{A} : \kappa(\mathcal{A}_i) = k_j\}$ contains all analyses sharing the same type-topic key k_j. Within each group, we aggregate the diagnostic attributes to form a collective "error profile" $\Psi_j = (\mathcal{E}_j, \mathcal{R}_j, \mathcal{F}_j)$, comprising the set of observed error types $\mathcal{E}_j = \{\epsilon_i : \mathcal{A}_i \in G_j\}$, recurring root causes $\mathcal{R}_j = \{\rho_i : \mathcal{A}_i \in G_j\}$, and common difficulty factors $\mathcal{F}_j$.

This aggregation transforms sparse, per-instance observations into dense, group-level patterns. By clustering failures with shared characteristics, the allocation phase enables the subsequent system to generate high-level guidance that addresses classes of errors simultaneously, ensuring that the enhanced prompts are both targeted and scalable.

3.3 Enhancement Generation

The final phase synthesizes the group-level error patterns into prompt enhancements that guide the model toward correct reasoning. For each group $G_j \in \mathcal{G}$, we first perform pattern analysis to extract actionable guidance, then integrate this guidance into the original prompt template.

Given a group G_j with its aggregated error profile Ψ_j, we query the analyzer model $\mathcal{M}_\phi$ to synthesize targeted remediation strategies. The analyzer examines the common error types $\mathcal{E}_j$, shared root causes $\mathcal{R}_j$, and recurring mistakes within the group, then produces structured guidance including typical pitfalls to avoid, verification steps to perform, and domain-specific reasoning strategies. This group-level synthesis captures patterns that may not be apparent from any single failure but emerge clearly when examining multiple related errors.

We define an enhancement generation function

$$\xi : \mathcal{G} \to \mathcal{S}^{(c)} \times \mathcal{S}^{(r)}, \quad \xi(G_j) = (E_j^{(c)}, E_j^{(r)}) \quad (4)$$

that produces two distinct enhancement variants to accommodate different reasoning scenarios. The "concise" enhancement $E_j^{(c)} \in \mathcal{S}^{(c)}$ provides brief warnings and key points, suitable for quick reference during inference. The "reasoning" enhancement $E_j^{(r)} \in \mathcal{S}^{(r)}$ supplies minimal hints designed to trigger self-correction without over-constraining the reasoning process. Formally, for each group G_j, we produce an enhancement pair $\xi(G_j) = (E_j^{(c)}, E_j^{(r)})$.

Category	Description	Typical Manifestation
Conceptual Misunderstanding	Fundamental confusion about domain principles	Misapplying laws; confusing related concepts
Calculation Error	Computational or mathematical mistakes	Arithmetic errors; unit conversion mistakes
Misreading	Misinterpretation of question or choices	Overlooking negation; misidentifying the question
Incomplete Analysis	Premature termination of reasoning	Stopping after partial solution
Wrong Elimination	Incorrect rejection of candidate answers	Eliminating correct answer based on flawed logic
Knowledge Gap	Absence of requisite domain knowledge	Missing facts; unfamiliar terminology

Table 2: Error categories and descriptions.

Benchmark	Focus	Categories		Details	
		Primary Domains	Sub-categories	Key Features	Size
DROP (Dua et al., 2019)	Discrete Reasoning	Numerical Operations	Add/Sub, Min/Max, Count, Select, Compare	NFL & History passages	96K
MGSM (Shi et al., 2023)	Multilingual Math	Grade-school Math	Arithmetic, Word Problems, Algebra	11 languages (BN, SW, TE incl.)	250
MMLU (Hendrycks et al., 2021)	General Knowledge	STEM Humanities Social Sciences	Physics, Chemistry, CS, Math, Biology History, Philosophy, Law, Ethics Economics, Psychology, Politics	57 subjects; Elem.–Prof.	15.9K
GPQA (Rein et al., 2024)	Graduate Science	Biology Physics Chemistry	Molecular Bio (8%), Genetics General (10%), Electromagnetism Organic (36%), General	PhD-level; Google-proof	448
HLE (Phan et al., 2025)	Expert Academic	Mathematics (41%) Sciences (27%) Tech & Humanities (32%)	Algebra, Analysis, Combinatorics Physics, Chemistry, Biology CS/AI, Engineering, Social Sci.	100+ areas; 14% multimodal	2.5K
Omni-MATH (Gao et al., 2024)	Olympiad Math	Algebra & Number Theory Geometry Analysis & Discrete	Linear, Abstract, Primes, Modular Euclidean, Analytic, Projective Calculus, Combinatorics, Graph Theory	33+ sub-domains; 10 levels	4.4K

Table 3: Question categories across six LLM evaluation benchmarks used for category-based hybrid enhancement.

The final enhanced prompt P' is constructed by appending the relevant enhancements to the base prompt P. We define an aggregation operator $\bigoplus$ that combines enhancements weighted by group cardinality $|G_j|$, prioritizing guidance derived from larger groups where more failures share the same type-topic characteristics. The resulting enhanced prompts are given by

$$P'^{(c)} = P \oplus \bigoplus_{j=1}^{m} w_j E_j^{(c)}, \quad P'^{(r)} = P \oplus \bigoplus_{j=1}^{m} w_j E_j^{(r)} \quad (5)$$

, where $w_j \propto |G_j|$, each embedding the collective remediation knowledge extracted from the failure analysis pipeline.

Beyond applying enhancements uniformly, we introduce a **Hybrid** strategy that dynamically selects the optimal enhancement type per question category. We apply four strategies: Concise, Reasoning, Concise+Reasoning, and Hybrid. The first three serve as ablation baselines. For the Hybrid strategy, each dataset is partitioned into train, validation, and test splits (8:1:1). The training set generates enhancements via Phases 1-3. The validation set determines which enhancement type performs best for each category c:

$$E_c^* = \underset{E \in \{E^{(c)}, E^{(r)}, E^{(c+r)}\}}{\arg\max} \text{Acc}(E, \mathcal{V}_c) \quad (6)$$

where $\mathcal{V}_c$ denotes validation questions in category c and $E^{(c+r)}$ combines both enhancement types. The selected E_c^* is applied to matching test questions.

4 Experiments

Datasets. We evaluate on six benchmarks spanning reasoning capacity and knowledge coverage. For reasoning capacity, we use DROP (Dua et al., 2019), a reading comprehension benchmark requiring discrete reasoning operations such as addition, counting, and sorting; MGSM (Shi et al., 2023), the Multilingual Grade School Math benchmark containing 250 problems; and OMNI-math (Gao et al., 2024), an Olympiad-level benchmark with 4,428 competition problems across 33 sub-domains. For knowledge coverage, we use MMLU (Hendrycks et al., 2021), which spans 57 subjects including STEM, humanities, and social sciences; GPQA (Rein et al., 2024), a graduate-level "Google-proof" Q&A benchmark with expert-written questions in biology, physics, and chemistry; and HLE (Phan et al., 2025) (Humanity's Last Exam), a frontier benchmark with 2,500 expert-level questions designed to test the limits of current AI systems.

For DROP, MGSM, MMLU, and GPQA, we use `gpt-3.5-turbo` for comparison with baselines. For the more challenging benchmarks, Omni-MATH and Humanity's Last Exam, we use `gpt-4o` due to its stronger reasoning capabilities and more recent training data cutoff.

Method	Enhancement	Reasoning		Knowledge	
		DROP	MGSM	MMLU	GPQA
MetaAgentSearch (Hu et al., 2025b)	-	79.4†	53.4†	69.6†	34.6†
Gödel Agent (Yin et al., 2025)	-	80.9†	64.2†	70.9†	34.9†
Zero-shot (Brown et al., 2020)	n/a	62.0	35.0	64.0	11.8
	Concise	63.5$_{+1.5}$	37.6$_{+2.6}$	60.7$_{-3.3}$	11.8$_{+0.0}$
	Reasoning	65.2$_{+3.2}$	37.0$_{+2.0}$	64.0$_{+0.0}$	12.7$_{+0.9}$
	Concise+Reasoning	63.8$_{+1.8}$	35.2$_{+0.2}$	64.2$_{+0.2}$	12.7$_{+0.9}$
	Hybrid	68.4$_{+6.4}$	39.4$_{+4.4}$	65.1$_{+1.1}$	20.0$_{+8.2}$
Zero-shot-CoT (Kojima et al., 2022)	n/a	74.5	52.9	65.8	16.4
	Concise	77.2$_{+2.7}$	54.4$_{+1.5}$	66.8$_{+1.0}$	19.1$_{+2.7}$
	Reasoning	78.8$_{+4.3}$	54.1$_{+1.2}$	69.0$_{+3.2}$	19.1$_{+2.7}$
	Concise+Reasoning	78.1$_{+3.6}$	54.6$_{+1.7}$	69.0$_{+3.2}$	17.3$_{+0.9}$
	Hybrid	**81.6**$_{+7.1}$	56.4$_{+3.5}$	70.5$_{+4.7}$	22.2$_{+5.8}$
Self-Refine (Madaan et al., 2023)	n/a	77.8	57.5	48.8	**36.4**
	Concise	80.5$_{+2.7}$	60.5$_{+3.0}$	60.5$_{+11.7}$	**38.2**$_{+1.8}$
	Reasoning	**82.1**$_{+4.3}$	58.4$_{+0.9}$	59.0$_{+10.2}$	**40.9**$_{+4.5}$
	Concise+Reasoning	**81.2**$_{+3.4}$	58.7$_{+1.2}$	63.9$_{+15.1}$	32.7$_{-3.7}$
	Hybrid	**84.3**$_{+6.5}$	61.3$_{+3.8}$	64.6$_{+15.8}$	**49.1**$_{+12.7}$
Self-Consistency (Wang et al., 2023)	n/a	79.5	63.5	61.8	18.2
	Concise	**83.8**$_{+4.3}$	**73.7**$_{+10.2}$	69.3$_{+7.5}$	26.4$_{+8.2}$
	Reasoning	**84.5**$_{+5.0}$	**73.2**$_{+9.7}$	**71.3**$_{+9.5}$	24.6$_{+6.4}$
	Concise+Reasoning	**83.2**$_{+3.7}$	**73.7**$_{+10.2}$	**71.7**$_{+9.9}$	21.8$_{+3.6}$
	Hybrid	**86.2**$_{+6.7}$	**74.3**$_{+10.8}$	**72.5**$_{+10.7}$	33.6$_{+15.4}$

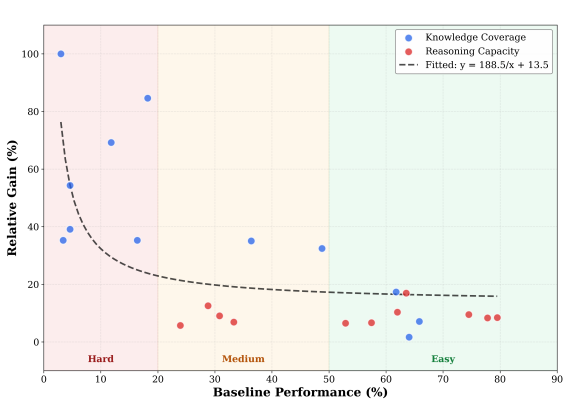
Table 4: Results on DROP, MGSM, MMLU, and GPQA benchmarks. DROP uses F1 score; others use accuracy (%). Subscripts indicate improvement over n/a baseline. **Bold** indicates results exceeding Gödel Agent. † are results obtained from Yin et al., 2025.

Implementation. We evaluate four base prompting methods: Zero-shot (Brown et al., 2020), Zero-shot-CoT (Kojima et al., 2022) which appends "Let's think step by step" to elicit reasoning, Self-Refine (Madaan et al., 2023) which iteratively critiques and improves responses, and Self-Consistency (Wang et al., 2023) which samples multiple reasoning paths and selects answers via majority voting.

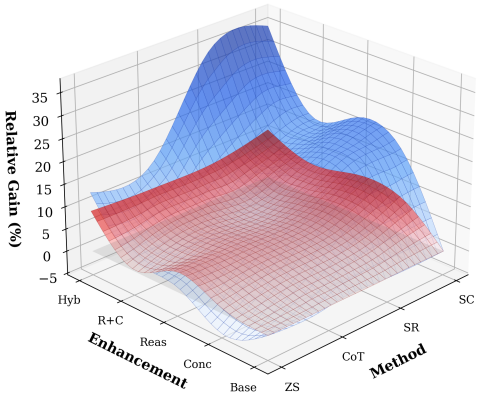
We apply four enhancement strategies to each prompting method: Concise (principle-based do's and don'ts for avoiding common errors), Reasoning (explicit step-by-step instructions to follow correct rationale), Concise+Reasoning (combining both), and Hybrid (dynamically selecting strategies based on question category). Concise, Reasoning, and Concise+Reasoning serve as ablation baselines to isolate the contribution of each enhancement type. For the Hybrid strategy, we leverage the question categories outlined in Table 3: DROP (5 discrete reasoning types), MGSM (3 math types × 11 languages), MMLU (57 subjects across 4 domains), GPQA (3 scientific domains), Humanity's Last Exam (8 broad categories), and Omni-MATH (33+ mathematical sub-domains). Each dataset is split into train:val:test = 8:1:1. The validation set is used to discover the optimal enhancement strategy for each question category, which is then applied to the corresponding categories in the test set. We compare against MategAgentSearch and Gödel Agent (Yin et al., 2025), in which Gödel Agent represents a state-of-the-art meta-learning optimized agent system. All experiments use temperature $T = 0$ and maximum token length of 3,000. For Self-Consistency, we sample $n = 5$ responses with temperature $T = 0.7$. We report F1 score for DROP and accuracy for all other benchmarks.

5 Results and Analysis

Main Results Table 4 presents results across four benchmarks: reasoning capacity (DROP and MGSM) and knowledge coverage (MMLU and GPQA). We compare our enhancement strategies against MetaAgentSearch (Hu et al., 2025b) and Gödel Agent (Yin et al., 2025). MetaAgentSearch and Gödel Agent employ instruction-free self-improvement through multiple recursive iterations, automatically discovering agent architectures without human-crafted guidance. In contrast, methods like Self-Refine (Madaan et al., 2023) rely on



(a) Baseline performance vs. relative gain from hybrid enhancement. Background shading indicates difficulty tiers.



(b) 3D surface of relative performance gain across prompting methods (X), enhancement types (Y), and accuracy (Z).

Figure 3: Performance gain analysis. (a) Inverse relationship between task difficulty and enhancement effectiveness. (b) Performance landscape comparison between Knowledge Coverage (blue) and Reasoning Capacity (red).

Method	Enhancement	Reasoning	Knowledge
		OMNI-math	HLE
Zero-shot	n/a	23.93	3.40
	Concise	23.44$_{-0.49}$	3.20$_{-0.20}$
	Reasoning	24.56$_{+0.63}$	3.40$_{+0.00}$
	R+C ⋆	22.43$_{-1.50}$	4.20$_{+0.80}$
	Hybrid	25.30$_{+1.37}$	4.60$_{+1.20}$
Zero-shot-CoT	n/a	30.81	4.60
	Concise	31.04$_{+0.23}$	5.10$_{+0.50}$
	Reasoning	31.83$_{+1.02}$	5.40$_{+0.80}$
	R+C ⋆	31.72$_{+0.91}$	5.20$_{+0.60}$
	Hybrid	33.60$_{+2.79}$	6.40$_{+1.80}$
Self-Refine	n/a	28.78	4.60
	Concise	28.67$_{-0.11}$	6.40$_{+1.80}$
	Reasoning	30.59$_{+1.81}$	5.50$_{+0.90}$
	R+C ⋆	29.12$_{+0.34}$	6.00$_{+1.40}$
	Hybrid	32.40$_{+3.62}$	**7.10**$_{+2.50}$
Self-Consistency	n/a	33.30	3.00
	Concise	33.60$_{+0.30}$	3.40$_{+0.40}$
	Reasoning	34.60$_{+1.30}$	5.20$_{+2.20}$
	R+C ⋆	32.50$_{-0.80}$	4.80$_{+1.80}$
	Hybrid	**35.60**$_{+2.30}$	6.00$_{+3.00}$

Table 5: Results on OMNI-math and HLE. Accuracy (%) reported. Subscripts indicate improvement over baseline. **Bold** indicates best result per dataset. ⋆s are short for reasoning plus concise enhancements

human-crafted prompts encoding explicit refinement strategies without groundtruth. Notably, even without our enhancements, Self-Refine (36.4%) already outperforms both Gödel Agent (34.9%) and MetaAgentSearch (34.6%) on GPQA. This demonstrates that well-designed human-crafted prompting can surpass instruction-free recursive optimization. With hybrid enhancement, Self-Consistency surpasses Gödel Agent on three benchmarks (DROP: 86.2 vs. 80.9, MGSM: 74.3 vs. 64.2, MMLU: 72.5 vs. 70.9), while Self-Refine with hybrid reaches 49.1% on GPQA. These results suggest that MARS offers a cost-effective alternative to complex recursive agent systems.

Zero-shot and Zero-shot-CoT show consistent improvements, with hybrid yielding +6.4 and +7.1 on DROP respectively. Self-Refine achieves dramatic gains on knowledge benchmarks: +15.8 on MMLU and +12.7 on GPQA. Self-Consistency benefits substantially across all benchmarks (+10.8 on MGSM, +10.7 on MMLU), indicating synergy between multiple reasoning paths and enhanced prompts. Hybrid enhancement consistently yields the largest improvements, validating category-aware selection. Reasoning generally outperforms Concise on reasoning-intensive tasks. Combining both (Concise+Reasoning) does not always yield additive benefits—on GPQA with Self-Refine, it underperforms individual enhancements (32.7% vs. 40.9% for Reasoning), suggesting interference when naively combining strategies.

Generalization to Challenging Benchmarks To investigate whether MARS generalizes to more challenging evaluation settings, we evaluate on two additional benchmarks: Omni-MATH and Humanity's Last Exam (HLE). Table 5 presents these results.

Omni-MATH tests advanced mathematical reasoning with baseline accuracies below 35%. Despite this difficulty, our enhancements provide consistent improvements. Reasoning enhancement proves particularly effective, yielding gains across all prompting methods (+0.63 for Zero-shot, +1.02 for Zero-shot-CoT, +1.81 for Self-Refine, +1.30 for Self-Consistency). Hybrid enhancement achieves

the best results, with Self-Consistency reaching 35.60% (+2.30). Notably, Concise+Reasoning often underperforms individual enhancements (e.g., 32.50% vs. 34.60% for Self-Consistency), consistent with the interference pattern observed in main results.

HLE represents an extremely challenging benchmark with baseline accuracies below 5%. Even in this difficult regime where models operate near floor-level performance, MARS demonstrates effectiveness. Self-Refine with hybrid enhancement achieves 7.10%, a 54.3% relative improvement over baseline (4.60%). Zero-shot-CoT with hybrid reaches 6.40% (+1.80), and Self-Consistency with Reasoning achieves 5.20% (+2.20). These gains indicate that our category-aware enhancements extract meaningful improvements even on problems designed to challenge state-of-the-art systems. Finally, using a different model for enhancement generation did not result in significant changes, as presented in Appendix D. Qwen2.5-72B-Instruct-Turbo produced comparable enhancement patterns across both knowledge coverage and reasoning capacity benchmark that confirms MARS enhancement's effectiveness generalizes across enhancement generators rather than being dependent on a specific model.

Performance Gain Analysis We analyze the relationship between baseline performance and relative gain from prompt enhancement using scatter plots and 3D surface visualizations across knowledge coverage (HLE, GPQA, MMLU) and reasoning capacity (OMNI-math, MGSM, DROP) benchmarks (Figure 3).

A significant inverse correlation exists between baseline performance and relative gain (Spearman $\rho = -0.654$, $p < 0.001$), with the fitted model gain $= 188.54/\text{baseline} + 13.48$ ($R^2 = 0.443$). Critically, this relationship is category-dependent: knowledge coverage datasets exhibit strong correlation ($\rho = -0.795$, $p = 0.002$), while reasoning capacity datasets show no significant relationship ($\rho = 0.264$, $p = 0.433$). This divergence suggests fundamentally different enhancement mechanisms—knowledge tasks benefit disproportionately at low baselines, whereas reasoning gains remain uniform across difficulty levels.

The 3D surface analysis reveals a method-enhancement interaction specific to reasoning tasks. For reasoning datasets, self-consistency combined with one-turn MARS enhancement produces significantly amplified gains compared to other prompting methods (7.31% vs. 2.66%; Mann-Whitney U, $p = 0.050$). This amplification pattern is *not observed* in knowledge coverage datasets, where high variance ($\sigma = 21.28\%$) obscures potential interaction effects and gains are primarily explained by the baseline-gain relationship.

Items	Statistic	p-value
Overall baseline-gain correlation	$\rho = -0.654$	< 0.001***
Knowledge baseline-gain correlation	$\rho = -0.795$	$= 0.002$**
Reasoning baseline-gain correlation	$\rho = 0.264$	$= 0.433$
SC amplification (Reasoning)	U	$= 0.050$*
Categories differ in gain	U	$= 0.003$**

Table 6: Statistical significance summary for gain analysis.

These findings suggest category-aware enhancement strategies: for knowledge tasks, prioritize low-baseline scenarios where gains are maximized; for reasoning tasks, leverage self-consistency methods which uniquely amplify MARS enhancement effects.

6 Conclusion

We presented MARS, a metacognitive framework that integrates principle-based reflection (learning what to avoid) with procedural reflection (learning how to succeed) for efficient self-improvement in LLM agents. Existing instruction-free self-improvement methods rely on multi-turn recursive optimization that is both computationally expensive and often underperforming. MARS overcomes both shortcomings by consolidating learning into a single recurrence cycle through structured summarization, while generating targeted, category-aware enhancements from systematic failure analysis. Experiments across six benchmarks demonstrate that MARS consistently outperforms state-of-the-art self-evolving systems with significantly reduced computational overhead, suggesting that human-inspired learning paradigms offer a practical alternative to resource-intensive recursive self-improvement.

Limitations

Two main limitations warrant consideration. First, the predefined error taxonomy may not generalize to all task types. Our six error categories and type-topic groupings are designed for structured benchmarks with clear correct answers. For open-ended or creative tasks where errors are less cate-

gorical and more subjective, these classifications may prove insufficient. Extending MARS to such domains would require developing more flexible categorization schemes.

Second, single-cycle learning trades depth for efficiency. While our approach significantly reduces computational cost compared to recursive methods, it inherently limits the improvement achievable in one pass. For applications prioritizing maximum performance over efficiency, we recommend applying MARS iteratively—using enhanced prompts to generate new failure sets, then deriving further refinements from residual errors.

References

John R. Anderson. 1983. *The Architecture of Cognition*. Harvard University Press, Cambridge, MA.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL-HLT*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*, pages 1126–1135.

John H. Flavell. 1979. Metacognition and cognitive monitoring: A new area of cognitive-developmental inquiry. *American Psychologist*, 34(10):906–911.

Bofei Gao and 1 others. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *ICLR*.

James Hiebert and Patricia Lefevre. 1986. Conceptual and procedural knowledge in mathematics: An introductory analysis. In *Conceptual and Procedural Knowledge: The Case of Mathematics*, pages 1–27. Lawrence Erlbaum Associates, Hillsdale, NJ.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Ajo Storkey. 2022. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5149–5169.

Shengran Hu, Cong Lu, and Jeff Clune. 2025a. Automated design of agentic systems. In *International Conference on Learning Representations*.

Shengran Hu, Cong Lu, and Jeff Clune. 2025b. Automated design of agentic systems. *Preprint*, arXiv:2408.08435.

Matthew Kaplan, Naomi Silver, Danielle LaVaque-Manty, and Deborah Meizlish. 2013. *Using Reflection and Metacognition to Improve Student Learning: Across the Disciplines, Across the Academy*. Stylus Publishing, Sterling, VA.

Manu Kapur. 2010. Productive failure in mathematical problem solving. *Instructional Science*, 38(6):523–550.

Manu Kapur. 2014. Productive failure in learning math. *Cognitive Science*, 38(5):1008–1022.

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*, volume 35, pages 22199–22213.

David A. Kolb. 1984. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall, Englewood Cliffs, NJ.

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 *Advances in Neural Information Processing Systems*, volume 36.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, and 317 others. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. 2024. Recursive introspection: Teaching language model agents how to self-improve. In *Advances in Neural Information Processing Systems*, volume 37.

Qwen Team. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. *COLM*.

Bethany Rittle-Johnson, Robert S. Siegler, and Martha Wagner Alibali. 2001. Developing conceptual understanding and procedural skill in mathematics: An iterative process. *Journal of Educational Psychology*, 93(2):346–362.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. 2025. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*. ICLR 2025 Workshop on Scaling Self-Improving Foundation Models.

Jürgen Schmidhuber. 2007. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial General Intelligence*, pages 199–226. Springer, Berlin, Heidelberg.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2025. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *ICLR*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Julie Dangremond Stanton, Amanda J. Sebesta, and John Dunlosky. 2021. Fostering metacognition to support student learning and performance. *CBE—Life Sciences Education*, 20(2):fe3.

Bas R. Steunebrink and Jürgen Schmidhuber. 2011. A family of gödel machine implementations. In *International Conference on Artificial General Intelligence*, pages 275–280. Springer.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6).

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

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. *Advances in Neural Information Processing Systems*, 35.

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

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. 2025. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27890–27913, Vienna, Austria. Association for Computational Linguistics.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2025. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5348–5375, Bangkok, Thailand. Association for Computational Linguistics.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

A Algorithm

Algorithm 1 presents the main MARS pipeline, which takes a set of failed questions and produces enhanced prompts.

Table 7 summarizes the key notation used in the algorithms.

Symbol	Description
$\mathcal{Q}$	Failed question set
$\mathbb{A}$	Set of failure analyses
$\mathcal{G}$	Partition of clustered failures
$\mathcal{M}_\phi$	Analyzer model
$\mathcal{E}$	Error taxonomy
τ	Question type
$\mathcal{T}$	Topic set
ϵ	Error type
ρ	Root cause
μ	Specific mistake
$E^{(c)}$	Concise enhancement
$E^{(s)}$	Specific enhancement
$E^{(r)}$	Reasoning enhancement
$\oplus$	Prompt concatenation

Table 7: Notation used in MARS algorithms.

B Enhancement Variants

Our method generates three enhancement variants for each type-topic group, each serving a distinct

Algorithm 1 MARS Enhancement Pipeline

Require: Failed questions $\mathcal{Q} = \{q_1, \ldots, q_n\}$, ground truths $\{a_i^*\}$, predictions $\{\hat{a}_i\}$, base prompt P, analyzer $\mathcal{M}_\phi$, error taxonomy $\mathcal{E}$

Ensure: Enhanced prompts $(P'^{(c)}, P'^{(s)}, P'^{(r)})$

▷ Phase 1: Evaluation
$\mathbb{A} \leftarrow \emptyset$
for $q_i \in \mathcal{Q}$ **do**
  $\mathcal{A}_i \leftarrow \textsc{Diagnose}(\mathcal{M}_\phi, q_i, a_i^*, \hat{a}_i)$
  $\mathbb{A} \leftarrow \mathbb{A} \cup \{\mathcal{A}_i\}$

▷ Phase 2: Failure Allocation
$\mathcal{G} \leftarrow \textsc{Cluster}(\mathbb{A})$

▷ Phase 3: Enhancement Generation
$\mathbb{E} \leftarrow \emptyset$
for $G_j \in \mathcal{G}$ **do**
  $(E_j^{(c)}, E_j^{(s)}, E_j^{(r)}) \leftarrow \textsc{Synthesize}(\mathcal{M}_\phi, G_j)$
  $\mathbb{E} \leftarrow \mathbb{E} \cup \{(E_j^{(c)}, E_j^{(s)}, E_j^{(r)})\}$

▷ Aggregate into final prompts
$(P'^{(c)}, P'^{(s)}, P'^{(r)}) \leftarrow \textsc{Aggregate}(P, \mathbb{E}, \mathcal{G})$

return $(P'^{(c)}, P'^{(s)}, P'^{(r)})$

purpose during inference. Table 8 summarizes their characteristics.

The **concise** variant $E^{(c)}$ provides brief warnings derived from common mistakes, suitable for scenarios where inference cost is a concern. The **specific** variant $E^{(s)}$ includes detailed verification steps and explicit reasoning strategies, appropriate when accuracy is prioritized over efficiency. The **reasoning** variant $E^{(r)}$ offers minimal guidance designed to trigger self-correction without over-constraining the model's reasoning process.

C Prompts

This appendix provides the complete prompt templates used in our experiments. All prompts use a structured XML-style output format with `<reasoning>` and `<answer>` tags to facilitate consistent response parsing. The placeholder `{question}` is replaced with the specific problem instance at inference time.

C.1 Zero-Shot Prompting

The zero-shot baseline provides the question directly without any demonstrations or reasoning instructions.

C.2 Zero-Shot Chain-of-Thought

Zero-shot chain-of-thought prompting (Kojima et al., 2022) elicits step-by-step reasoning without providing exemplars.

C.3 Few-Shot Chain-of-Thought

Few-shot chain-of-thought prompting (Wei et al., 2022) provides exemplars demonstrating the reasoning process.

C.4 Self-Consistency

Self-consistency (Wang et al., 2023) samples multiple reasoning paths and aggregates answers via majority voting. We evaluate configuration with 10 samples.

C.5 Self-Refine

Self-refine (Madaan et al., 2023) enables iterative improvement of responses through self-feedback.

C.6 Enhancement Analyzer

The Enhancement Analyzer is the first LLM-powered agent in our zero-shot enhancement pipeline. It performs individual failure analysis by examining each incorrectly answered GPQA question to determine the precise cause of error. Given a failed question along with the model's predicted answer and reasoning, the analyzer classifies the question type (factual, conceptual, calculation, application, analysis, or comparison), identifies specific scientific topics, and diagnoses the error type (e.g., conceptual misunderstanding, calculation error, misreading, incomplete analysis, wrong elimination, or knowledge gap). The agent produces a structured JSON output containing the root cause explanation, the specific reasoning step that failed, required domain knowledge, and factors contributing to the question's difficulty. This granular analysis enables downstream pattern recognition across multiple failures.

C.7 Enhancement Synthesizer

The Enhancement Synthesizer is the second LLM-powered agent that operates on grouped failures sharing common characteristics. After the Enhancement Analyzer processes individual questions, failures are clustered by question type and topic. The Synthesizer then analyzes each cluster to identify recurring error patterns, synthesize shared root causes, and generate targeted enhancement strategies. For each type-topic group, it produces common mistake patterns, critical warnings, verification steps, topic-specific guidance, and a concise prompt addition designed to prevent similar errors. This synthesized knowledge is then used to construct three variants of enhanced prompts (concise, specific, and reasoning-focused), each tailored to

Variant	Purpose	Content	Length
Concise ($E^{(c)}$)	Quick reference	Warnings and key points	Short
Specific ($E^{(s)}$)	Thorough guidance	Mistakes, verification steps, approach	Detailed
Reasoning ($E^{(r)}$)	Self-correction	Minimal hints for discovery	Minimal

Table 8: Comparison of three enhancement variants.

Zero-Shot Prompt

```
Answer the following question.

Question: {question}
```

Sampling Parameters: Temperature = 0, Samples = 1

Figure 4: Zero-shot prompt template.

address the identified weaknesses while maintaining the base prompting strategy's structure.

C.8 Model Configuration

Table 9 summarizes the model configuration used across all experiments.

Parameter	Value
Default Model	GPT-4o
Evaluation Model	o3-mini
Max Tokens	2000
Timeout	30 seconds

Table 9: Model configuration settings.

Component	Concise	Reasoning	Specific
Explicit warnings	Yes	No	Yes
Action sequences	Yes	No	Yes
Process guidance	Brief	Detailed	Detailed
Verification steps	No	Yes	Yes
Approach description	No	Yes	Yes
Relative length	Short	Medium	Long

Table 10: Structural comparison of MARS enhancement types.

D Enhancement Generation with Open-source Model

To evaluate the generalizability of MARS enhancements across different enhancement generators, we replicate experiments using Qwen2.5-72B-Instruct-Turbo (Qwen Team, 2024) for enhancement generation on one knowledge coverage dataset (GPQA) and one reasoning capacity dataset (OMNI-math). Results are presented in Tables 11 and 12.

Analysis. Qwen-generated enhancements demonstrate consistent improvement patterns across both datasets. The hybrid enhancement achieves the highest gains, with self-refine + hybrid reaching 48.20% on GPQA (a 32.6% relative improvement over baseline). Individual enhancements show modest gains: concise enhancement improves zero-shot by 7.7% relative, while reasoning+concise provides the largest single-enhancement gain for self-consistency (24.9% relative). These results confirm that MARS enhancement effectiveness generalizes across enhancement generators, though optimal enhancement selection remains task-dependent.

E One-Turn MARS Enhancement Samples

MARS generates three types of one-turn prompt enhancements from the same error analysis, each with different structures and verbosity levels. We demonstrate each type using the Algebra_Equations category under zero-shot prompting.

E.1 Concise Enhancement

The concise enhancement (Figure 11) provides compact, action-oriented guidance organized by type-topic groups. Each group includes warning indicators (`[!]`) highlighting common failure patterns with failure counts, followed by action arrows (`->`) specifying recommended problem-solving procedures. This format minimizes prompt length while preserving critical guidance.

Zero-Shot Chain-of-Thought Prompt

```
Solve this problem step by step.
Question: {question}

Provide your response in the following format:

<reasoning>
[Show your step-by-step work and explain your thinking process here]
</reasoning>

<answer>
[Provide only the final answer here]
</answer>
```

Sampling Parameters: Temperature = 0, Samples = 1

Figure 5: Zero-shot chain-of-thought prompt template.

Few-Shot Chain-of-Thought Prompt

```
Here are some examples of how to solve similar problems:

Example 1:
Question: If x + 3 = 7, what is x?
<reasoning>
To find x, I need to isolate it on one side.
Subtracting 3 from both sides: x + 3 - 3 = 7 - 3
Simplifying: x = 4
</reasoning>
<answer>
4
</answer>

Now solve this problem:
Question: {question}

Provide your response in the following format:

<reasoning>
[Show your step-by-step work and explain your thinking process here]
</reasoning>

<answer>
[Provide only the final answer here]
</answer>
```

Sampling Parameters: Temperature = 0, Samples = 1

Figure 6: Few-shot chain-of-thought prompt template with one demonstration example.

E.2 Reasoning Enhancement

The reasoning enhancement (Figure 12) emphasizes problem-solving strategies over explicit warnings. Each type-topic group receives a bulleted consideration (*) describing the recommended reasoning approach. This format focuses on *how* to think about problems rather than *what* to avoid, making it effective for tasks requiring flexible reasoning.

E.3 Specific Enhancement (Reasoning + Concise)

The specific enhancement (Figure 13) combines both approaches into a comprehensive three-part structure for each type-topic group: (1) **Common Mistakes** (x) explicitly enumerate failure patterns; (2) **Verification Steps** (+) provide concrete validation actions; (3) **Approach** describes the recommended problem-solving methodology. This format maximizes guidance completeness at the cost of increased prompt length.

E.4 Enhancement Comparison

Table 10 summarizes the structural differences between enhancement types.

The hybrid enhancement strategy evaluates all three types on a validation set and selects the optimal enhancement for each question, combining their complementary strengths.

Self-Consistency Prompt (10 Samples)

```
Solve this problem step by step.
Question: {question}

Provide your response in the following format:

<reasoning>
[Show your step-by-step work and explain your thinking process here]
</reasoning>

<answer>
[Provide only the final answer here]
</answer>
```

Sampling Parameters: Temperature = 0.7, Samples = 10

Figure 7: Self-consistency prompt template (10 samples).

Self-Refine Prompt

```
Solve this problem step by step.
Question: {question}

Provide your response in the following format:

<reasoning>
[Show your step-by-step work and explain your thinking process here]
</reasoning>

<answer>
[Provide only the final answer here]
</answer>
```

Sampling Parameters: Temperature = 0, Samples = 1

Figure 8: Self-refine prompt template.

F Computational Cost Analysis

We compare the computational costs of MARS against recursive self-improvement baselines: Gödel Agent (Yin et al., 2025) and Meta Agent Search (ADAS) (Hu et al., 2025a). Costs are estimated based on reported experimental configurations and current API pricing.

F.1 Baseline Method Costs

Meta Agent Search. As reported in (Hu et al., 2025a), Meta Agent Search runs for **25 iterations**, where each iteration involves: (1) the meta agent (GPT-4) programming a new agent design, (2) self-reflection refinement (2 iterations per proposal, plus up to 3 error-correction refinements), and (3) evaluation on validation data using GPT-3.5. The authors report a total cost of approximately **$300** across four benchmarks (DROP, MGSM, MMLU, GPQA). This high cost stems from the extensive GPT-4 usage for iterative agent design and the growing archive of discovered agents that must be included in each subsequent prompt.

Gödel Agent. As reported in (Yin et al., 2025), Gödel Agent performs **30 recursive self-improvements** across four benchmarks, with a total cost of approximately **$15**. The framework uses GPT-4o for self-modification and GPT-3.5 for policy evaluation. The reduced cost compared to Meta Agent Search is attributed to continuous self-optimization that enables faster convergence. However, the authors note that the main cost driver is the continuously growing historical memory, suggesting that longer optimization runs would incur substantially higher costs.

F.2 MARS Cost Estimation

MARS operates in a **single recurrence cycle** with three phases:

1. **Evaluation Phase**: The analyzer model processes each failed question to produce structured diagnoses. For a typical benchmark with $\sim$200 failed questions, this requires $\sim$200 API calls.

2. **Failure Allocation Phase**: Pure computa-

Enhancement Analyzer Prompt

Analyze this failed GPQA (Graduate-Level Google-Proof Q&A) question. The model used "{strategy}" prompting strategy.

Domain: {domain}
Subdomain: {subdomain}

Question: {question}

Answer Choices:
{choices_text}

Correct Answer: {correct_letter} - {correct_answer}
Model's Answer: {predicted_letter} - {model_answer}

Model's Reasoning (excerpt): {reasoning}

Provide a comprehensive analysis in JSON format:

```
{
    "question_type": "<factual/conceptual/calculation/application/analysis/comparison>",
    "topics": ["<specific scientific topic 1>", "<topic 2>"],
    "error_type": "<conceptual_misunderstanding/calculation_error/misreading/...>",
    "root_cause": "<detailed explanation of why wrong answer was chosen>",
    "specific_mistake": "<exact reasoning step that failed>",
    "requires_knowledge": ["<scientific knowledge 1>", "<knowledge 2>"],
    "difficulty_factors": ["<what makes this challenging for AI>"]
}
```

Focus on why the model chose the wrong answer in this multiple-choice context.

Figure 9: Enhancement Analyzer prompt template for individual failure analysis.

Method	Baseline	Concise	Reasoning	R+C	Hybrid
Zero-shot	11.82	12.73	11.82	13.64	19.10
Zero-shot-CoT	16.40	18.18	15.45	13.64	22.70
Self-refine	36.36	37.27	39.09	37.27	48.20
Self-consistency	18.20	19.09	17.27	22.73	32.60

Table 11: GPQA performance (%) with Qwen-generated enhancements.

tional grouping with no API calls required.

3. **Enhancement Generation Phase**: The synthesizer generates enhancements for each type-topic group. With ∼15–20 groups per benchmark, this requires ∼40–60 API calls (including concise, reasoning, and specific variants).

Using GPT-3.5-turbo ($0.0005/1K input, $0.0015/1K output tokens) for both analysis and synthesis, the estimated cost per benchmark is:

F.3 Cost Comparison Summary

Table 14 summarizes the computational requirements across methods.

F.4 Analysis

The cost reduction achieved by MARS stems from three design decisions:

1. **Single-cycle learning**: While recursive methods require 25–30 iterations to converge, MARS consolidates learning into one pass, eliminating the multiplicative cost of iteration.

2. **No growing context**: Recursive methods accumulate historical memory (Gödel Agent) or an archive of discovered agents (Meta Agent Search), causing token usage to grow with each iteration. MARS processes fixed-size inputs throughout.

3. **Efficient model selection**: MARS uses GPT-3.5 for both analysis and synthesis, while re-

Enhancement Synthesizer Prompt

Analyze these {num_failures} failed GPQA questions that share:

> Domain: {domain}
> Question Type: {question_type}
> Topics: {topics}
> Prompting Strategy Used: {strategy}

Sample Failures:
{failures_summary}

Common Error Patterns: {error_patterns}
Required Knowledge: {required_knowledge}
Difficulty Factors: {difficulty_factors}

Create a targeted enhancement strategy in JSON format:

```
{
    "common_mistakes": ["<specific mistake pattern 1>", "<pattern 2>"],
    "key_warnings": ["<critical warning for this topic 1>", "<warning 2>"],
    "verification_steps": ["<verification step 1>", "<step 2>"],
    "topic_specific_guidance": "<detailed guidance for these topics>",
    "type_specific_approach": "<approach for this question type>",
    "enhanced_prompt_addition": "<concise 2-3 sentence prompt addition>"
}
```

Make it specific to {question_type} questions about {topics} in {domain}.

Figure 10: Enhancement Synthesizer prompt template for pattern-based enhancement generation.

Method	Baseline	Concise	Reasoning	R+C	Hybrid
Zero-shot	11.82	12.73	11.82	13.64	19.10
Zero-shot-CoT	16.40	18.18	15.45	13.64	22.70
Self-refine	36.36	37.27	39.09	37.27	48.20
Self-consistency	18.20	19.09	17.27	22.73	32.60

Table 12: OMNI-math performance (%) with Qwen-generated enhancements.

Phase	API Calls	Tokens (K)	Est. Cost
Evaluation	∼200	∼400	∼$0.40
Enhancement Gen.	∼50	∼150	∼$0.15
Total (per benchmark)	∼250	∼550	∼$0.55
Total (4 benchmarks)	∼1,000	∼2,200	∼$2.20

Table 13: Estimated MARS computational cost using GPT-3.5-turbo.

cursive methods require GPT-4/GPT-4o for meta-level reasoning. As shown in Appendix D, enhancement quality is robust to generator model choice.

The cost-performance trade-off favors MARS: at 136× lower cost than Meta Agent Search and 6.8× lower than Gödel Agent, MARS achieves comparable or superior performance (Table 4). For applications requiring maximum performance regardless of cost, MARS can be applied iteratively—using enhanced prompts to generate new failure sets for further refinement—while still maintaining substantial cost advantages over recursive baselines.

G MARS Implementation Details

This appendix provides key code snippets from the MARS implementation.

G.1 Data Structures

Listing 1 defines the core data structures corresponding to Equations 1–3.

```python
@dataclass
class IndividualFailureAnalysis:
    """Structured analysis A_i = (tau_i, T_i, epsilon_i, rho_i, mu_i)"""
    question_id: str
    question_text: str
```

Concise Enhancement Sample

```
Answer the following question.
Question: {question}

## SPECIALIZED GUIDANCE FOR ALGEBRA_EQUATIONS (ZERO_SHOT)
### Critical Warnings by Question Type:

**Calculation questions about algebra/number theory** (7 failures):
[!] Ensure that all algebraic relationships and constraints are clearly
    defined before attempting calculations.
[!] Be cautious of jumping to conclusions without verifying all potential
    solutions and constraints.
 -> When solving calculation problems in algebra and number theory,
    carefully define all variables and relationships before proceeding.
    Consider all possible integer solutions and verify constraints.

**Calculation questions about algebra/equations** (5 failures):
[!] Be cautious of prematurely concluding an answer without completing
    all necessary steps.
[!] Ensure all potential simplifications and substitutions are considered.
 -> Approach this problem by identifying any potential simplifications
    or substitutions. Carefully verify each algebraic manipulation step.

**Analysis questions about functional equations** (3 failures):
[!] Ensure all solutions satisfy the functional equation for all x and y.
 -> When solving functional equations, verify each proposed solution
    thoroughly. Explore transformations like symmetry or substitution.
                                                        ...
```

Enhancement Type: Warning Indicators [!] + Action Sequences (->)

Figure 11: Concise enhancement template structure. Warnings identify failure patterns; arrows provide actionable guidance.

Method	Iterations	Meta Model	Eval Model	Cost (4 benchmarks)	Relative
Meta Agent Search	25	GPT-4	GPT-3.5	∼$300	136×
Gödel Agent	30	GPT-4o	GPT-3.5	∼$15	6.8×
MARS (Ours)	**1**	GPT-3.5	GPT-3.5	∼$2.20	**1**×

Table 14: Computational cost comparison across self-improvement methods. Costs based on reported values (Yin et al., 2025; Hu et al., 2025a) and our estimates.

```python
    question_type: str      # tau_i in Y (question type)
    topics: List[str]       # T_i subset of D (topic set)
    error_type: str         # epsilon_i in E (error taxonomy)
    root_cause: str         # rho_i (reasoning deficit)
    specific_mistake: str   # mu_i (divergence point)
    requires_knowledge: List[str]
    difficulty_factors: List[str]

@dataclass
class QuestionTypeTopicGroup:
    """Group G_j with error profile Psi_j = (E_j, R_j, F_j)"""
    question_type: str
    topics: List[str]
    failures: List[IndividualFailureAnalysis]
    common_error_patterns: List[str]  # E_j
    shared_root_causes: List[str]     # R_j
    required_knowledge: Set[str]
    key_difficulty_factors: List[str] # F_j

@dataclass
class TypeTopicEnhancement:
    """Enhancement variants for type-topic group:
       E^(c): concise, E^(r): reasoning, E^(c+r): specific"""
    question_type: str
    topics: List[str]
    num_questions: int              # |G_j| for weight w_j
    # For concise enhancement E^(c):
    key_warnings: List[str]         # Warning indicators [!]
    # For reasoning enhancement E^(r):
    enhanced_prompt_addition: str   # Process-oriented
      guidance
    # For specific enhancement E^(c+r) = concise + reasoning:
    common_mistakes: List[str]      # Explicit error patterns
      (x)
    verification_steps: List[str]   # Validation actions (+)
    type_specific_approach: str     # Methodological guidance
```

Listing 1: Core data structures for MARS pipeline.

G.2 Phase 1: Individual Failure Analysis

Listing 2 shows the evaluation phase that produces structured analyses $\mathcal{A}_i$ for each failed question.

```python
def analyze_individual_failure(self, failure: Dict,
                               strategy: str) ->
    IndividualFailureAnalysis:
    question = failure.get('question', '')
    correct_answer = failure.get('correct_answer', '')
    model_answer = failure.get('predicted_answer', '')

    prompt = f"""Analyze this failed question using "{strategy
      }" strategy.
Question: {question[:2000]}
Correct: {correct_answer[:500]}
Model Answer: {model_answer[:500]}

```

Reasoning Enhancement Sample

```
Answer the following question.
Question: {question}

## SPECIALIZED GUIDANCE FOR ALGEBRA_EQUATIONS (ZERO_SHOT)
### Key Considerations by Problem Type:

* Calculation (algebra/number theory): When solving calculation problems
  in algebra and number theory, carefully define all variables and
  relationships before proceeding. Consider all possible integer
  solutions and verify that they satisfy the given constraints.

* Calculation (algebra/equations): Approach this problem by identifying
  any potential simplifications or substitutions that could ease
  calculations. Carefully verify each algebraic manipulation step.

* Analysis (functional equations/real analysis): When solving functional
  equations, verify each proposed solution thoroughly. Consider specific
  values of x and y, and explore transformations like symmetry.

* Reasoning (functional equations/real analysis): When tackling
  functional equations, consider starting with simple functions like
  linear or constant. Use substitutions to simplify the equation.

* Calculation (algebra/systems of equations): When solving algebraic
  systems of equations, first seek patterns or symmetries that might
  simplify the problem. Verify solutions by substituting them back.

* Calculation (algebra/optimization): When tackling algebraic
  optimization problems, carefully analyze the constraints and consider
  algebraic manipulations that simplify the expression.
```

Enhancement Type: Process-Oriented Guidance (*)

Figure 12: Reasoning enhancement template structure. Bullet points provide process-oriented guidance for each problem type.

Reasoning + Concise Enhancement Sample

```
Answer the following question.
Question: {question}

## SPECIALIZED GUIDANCE FOR ALGEBRA_EQUATIONS (ZERO_SHOT)
### Detailed Guidance by Question Type and Topic:

**Calculation - algebra & number theory** (7 failures):

Common Mistakes:
  x Failing to correctly establish algebraic relationships from word
    problems, leading to incorrect equation setups.
  x Misapplying integer constraints and failing to consider all possible
    integer solutions that satisfy the given conditions.
  x Jumping to conclusions without systematically verifying each
    candidate solution against all problem constraints.

Verification Steps:
  + Break down the problem into smaller, manageable parts and clearly
    define all variables and their relationships before solving.
  + Verify each step of the calculation, especially when dealing with
    integer constraints, by checking edge cases.
  + Ensure all solutions satisfy the original equations by substituting
    back and confirming consistency with problem conditions.

Approach: In calculation questions involving algebra and number theory,
start by translating the problem into mathematical expressions. Use
algebraic manipulation to simplify and solve these equations, and apply
number theory principles to handle integer constraints. Consider
multiple approaches such as substitution, factorization, or case
analysis to ensure comprehensive exploration of solutions.

**Calculation - algebra & equations** (5 failures):

Common Mistakes:
  x Failure to complete calculations, leading to incomplete answers.
  x Incorrect application of algebraic manipulations or simplifications.

Verification Steps:
  + Review the problem for potential simplifications or substitutions.
  + Verify each step of the algebraic manipulation to ensure no errors.
  + Cross-check final answer by substituting back into original equations.

Approach: For calculation questions in algebra, emphasize a step-by-step
approach that includes checking for simplifications and verifying each
manipulation. Leverage known algebraic identities or properties.
                                                        ...
```

Enhancement Type: Common Mistakes (x) + Verification Steps (+) + Approach

Figure 13: Specific enhancement template structure. Combines explicit mistake warnings, verification steps, and methodological guidance.

```
Provide JSON analysis:
{{
    "question_type": "<factual/conceptual/calculation/
      application>",
    "topics": ["<topic_1>", "<topic_2>"],
    "error_type": "<conceptual_misunderstanding/
      calculation_error/...>",
    "root_cause": "<fundamental reasoning deficit>",
    "specific_mistake": "<exact step where logic diverged>",
    "requires_knowledge": ["<knowledge_1>"],
    "difficulty_factors": ["<factor_1>"]
}}"""

    result = call_llm(self.client, self.model,
                      [{"role": "user", "content": prompt}],
                      temperature=0.3, max_tokens=800)
    data = json.loads(extract_json(result))

    return IndividualFailureAnalysis(
        question_type=data.get('question_type'),
        topics=data.get('topics', []),
        error_type=data.get('error_type'),
        root_cause=data.get('root_cause', ''),
        specific_mistake=data.get('specific_mistake', ''),
        requires_knowledge=data.get('requires_knowledge', []),
        difficulty_factors=data.get('difficulty_factors', []))
```

Listing 2: Code for individual failure analysis (Evaluation phase).

G.3 Phase 2: Type-Topic Grouping

Listing 3 implements the grouping function κ (Equation 2) that partitions analyses into groups $\mathcal{G} = \{G_j\}$.

```
def group_by_type_topic(self, analyses: List[
      IndividualFailureAnalysis]
                        ) -> List[QuestionTypeTopicGroup]:
    """Apply grouping function kappa: A -> Y x 2^D"""

    # Partition by composite key (tau_i, T_i)
    groups = defaultdict(list)
    for analysis in analyses:
        key = (analysis.question_type,
               frozenset(analysis.topics[:2]))
        groups[key].append(analysis)

    type_topic_groups = []
    for (q_type, topics), group_analyses in groups.items():
        # Aggregate into error profile Psi_j
        error_patterns = [a.error_type for a in group_analyses
      ]
        root_causes = [a.root_cause for a in group_analyses]
        required_knowledge = set()
        difficulty_factors = []

        for a in group_analyses:
            required_knowledge.update(a.requires_knowledge)
            difficulty_factors.extend(a.difficulty_factors)

        group = QuestionTypeTopicGroup(
            question_type=q_type,
            topics=list(topics),
            failures=group_analyses,
            common_error_patterns=list(set(error_patterns)),
            shared_root_causes=list(set(root_causes)),
            required_knowledge=required_knowledge,
            key_difficulty_factors=list(set(difficulty_factors
      )))
        type_topic_groups.append(group)

    return type_topic_groups
```

Listing 3: Code for failure allocation via type-topic grouping.

G.4 Phase 3: Enhancement Generation

Listing 4 implements the enhancement function ξ (Equation 4) and prompt aggregation (Equation 5). The three enhancement variants are: (1) **concise** $E^{(c)}$: warning indicators + action sequences; (2) **reasoning** $E^{(r)}$: process-oriented guidance; and (3) **specific** $E^{(c+r)}$: the combination of concise and reasoning, providing common mistakes, verification steps, and methodological approach.

```
def create_enhanced_prompts(self, base_prompt: str,
                            enhancements: List[
      TypeTopicEnhancement],
                            strategy: str, category: str) ->
      Dict[str, str]:
    """Generate P'^(c) and P'^(r) via weighted aggregation"""

    # Sort by |G_j| for weight w_j (Eq. 5)
    sorted_enh = sorted(enhancements,
                        key=lambda e: e.num_questions, reverse
      =True)
    all_prompts = {}

    # Concise enhancement E^(c): warnings + action sequences
    if 'concise' in self.enhancement_types:
        text = f"\n## GUIDANCE FOR {category.upper()}\n"
        text += "### Critical Warnings by Question Type:\n\n"
        for enh in sorted_enh[:8]:  # Top-weighted groups
            text += f"**{enh.question_type} ({'/'.join(enh.
      topics)})** "
            text += f"({enh.num_questions} failures):\n"
            text += "[!] " + " | ".join(enh.key_warnings[:3])
      + "\n"
            text += f" -> {enh.enhanced_prompt_addition}\n\n"
        all_prompts['concise'] = base_prompt + text  # P' = P
      + E

    # Reasoning enhancement E^(r): process-oriented hints
    if 'reasoning' in self.enhancement_types:
        text = f"\n## GUIDANCE FOR {category.upper()}\n"
        text += "### Key Considerations by Problem Type:\n\n"
        for enh in sorted_enh[:6]:
            text += f"* {enh.question_type} ({'/'.join(enh.
      topics)}): "
            text += f"{enh.enhanced_prompt_addition}\n"
        all_prompts['reasoning'] = base_prompt + text

    # Specific enhancement E^(c+r): combines concise +
      reasoning
    # Includes: mistakes (concise) + verification (concise) +
      approach (reasoning)
    if 'specific' in self.enhancement_types:
        text = f"\n## GUIDANCE FOR {category.upper()}\n"
        for enh in sorted_enh[:10]:
            text += f"**{enh.question_type} - {' & '.join(enh.
      topics)}**\n"
            # From concise: explicit warnings as mistake
      patterns
            text += "Common Mistakes:\n"
            for m in enh.common_mistakes[:3]: text += f"  x {m
      }\n"
            # From concise: action sequences as verification
            text += "Verification Steps:\n"
            for s in enh.verification_steps[:4]: text += f"  +
       {s}\n"
            # From reasoning: process-oriented approach
            text += f"Approach: {enh.type_specific_approach}\n
      \n"
        all_prompts['specific'] = base_prompt + text

    return all_prompts
```

Listing 4: Code for enhancement generation and prompt aggregation.

G.5 Hybrid Selection

Listing 5 implements the hybrid strategy (Equation 6) that selects optimal enhancement per category.

```
def select_hybrid_enhancement(self, val_data: Dict[str, List],
```

```
                              enhancements: Dict) -> Dict[str, str]:
    """Select E*_c = argmax Acc(E, V_c) for each category c where E in {E^(c), E^(r), E^(c+r)} (concise, reasoning, specific)"""

    optimal = {}
    # specific = concise + reasoning (c+r)
    etypes = ['concise', 'reasoning', 'specific']

    for category, val_questions in val_data.items():
        best_acc, best_type = 0, 'concise'

        for etype in etypes:
            enhanced_prompt = enhancements.get(f"{category}_{etype}")
            if not enhanced_prompt: continue

            correct = sum(1 for q in val_questions
                          if self.evaluate(enhanced_prompt, q)
                             == q['correct_answer'])
            accuracy = correct / len(val_questions)

            if accuracy > best_acc:
                best_acc, best_type = accuracy, etype

        optimal[category] = best_type
        print(f"  {category}: '{best_type}' (acc: {best_acc:.1%})")

    return optimal
```

Listing 5: Code for hybrid enhancement selection.