Reflection in the Dark: Exposing and Escaping the Black Box in Reflective Prompt Optimization

Shiyan Liu[1,2*], **Qifeng Xia**[2*], **Qiyun Xia**[3*], **Yisheng Liu**[2], **Xinyu Yu**[2], **Rui Qu**[2]
[1]University of California, Berkeley
[2]Huazhong University of Science and Technology
[3]Hefei University of Technology
shiyanliu@berkeley.edu, shyl@hust.edu.cn

Abstract

Automatic prompt optimization (APO) has emerged as a powerful paradigm for improving LLM performance without manual prompt engineering. Reflective APO methods such as GEPA iteratively refine prompts by diagnosing failure cases, but the optimization process remains black-box and label-free, leading to uninterpretable trajectories and systematic failure. We identify and empirically demonstrate four limitations: on GSM8K with a defective seed, GEPA degrades accuracy from 23.81% to 13.50%. We propose VISTA, a multi-agent APO framework that decouples hypothesis generation from prompt rewriting, enabling semantically labeled hypotheses, parallel mini-batch verification, and interpretable optimization trace. A two-layer explore-exploit mechanism combining random restart and epsilon-greedy sampling further escapes local optima. VISTA recovers accuracy to 87.57% on the same defective seed and consistently outperforms baselines across all conditions on GSM8K and AIME2025.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across diverse tasks (Brown et al., 2020; Wei et al., 2022; Kojima et al., 2022; Wang et al., 2023), yet their performance remains highly sensitive to prompt design: small changes in wording or instruction order can lead to dramatic differences in output quality. Manual prompt engineering is labor-intensive and requires extensive trial and error, motivating a growing body of work on Automatic Prompt Optimization (APO) (Ye et al., 2024; Chen et al., 2024; Sahoo et al., 2024).

APO addresses this bottleneck by iteratively refining prompts based on task feedback, replacing human intuition with algorithmic search. Early approaches such as OPRO (Yang et al., 2024) and ProTeGi (Pryzant et al., 2023) demonstrated that LLMs can serve as effective optimizers, proposing prompt edits in natural language guided by performance signals. More recently, reflective APO methods such as GEPA (Agrawal et al., 2026) have pushed this further by combining natural language reflection with genetic evolution and Pareto-based candidate selection.

Despite their promise, we identify a fundamental limitation shared by reflective APO methods: the optimization process is entirely black-box and label-free. Failure diagnosis and prompt rewriting are collapsed into a single reflection step, producing no record of what root cause was attributed, no semantic structure on the optimization trajectory, and no mechanism to detect when the search has been trapped from the start. Without this structure, the optimizer has no sense of where it started, what it can diagnose, where it has been, or whether its results will generalize. Figure 1 illustrates a representative failure. Concretely, GEPA with its

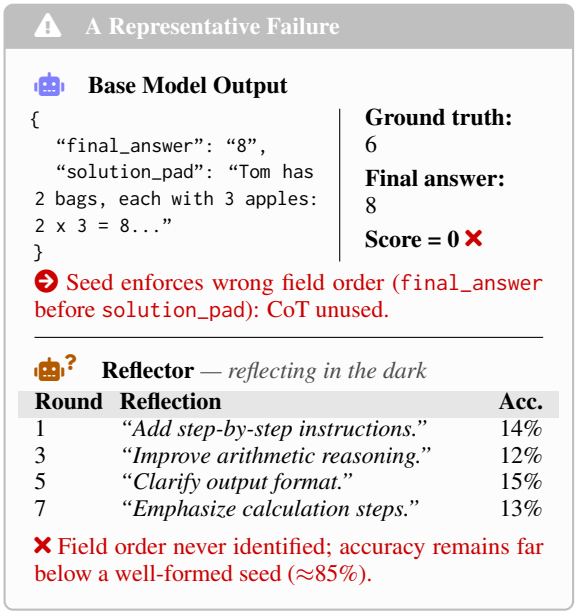


Figure 1: A representative failure of GEPA under the defective seed.

*Equal contribution.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), pages 76–109
July 2-7, 2026 ©2026 Association for Computational Linguistics

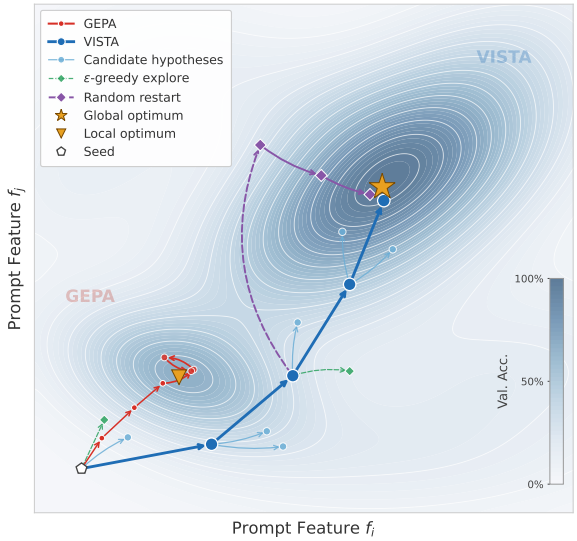


Figure 2: Conceptual illustration of optimization trajectories under the defective seed.

official, defective seed prompt on GSM8K (Cobbe et al., 2021) degrades accuracy from 23.81% to 13.50%, and the actual root cause remains unaddressed across all optimization rounds.

We formalize this as four systematic limitations that form a causal chain: **[L1] SEED TRAP**: optimization is sensitive to seed prompts, which silently constrain the search space and trap it in defective regions. **[L2] ATTRIBUTION BLINDSPOT**: the reflector's attribution space is doubly bounded—by its prior distribution and by its own capability—systematically missing root causes outside either bound. **[L3] TRAJECTORY OPACITY**: even when attribution points in the right direction, the optimization trajectory is entirely label-free, making the full evolution uninterpretable and preventing the optimizer from accumulating directional experience. **[L4] TRANSFER FRAGILITY**: optimized prompts are model-specific, failing silently when transferred across base models.

To address these limitations, we propose **VISTA** (**V**erifiable, **I**nterpretable, **S**emantic-**TrA**ce Prompt Optimization). The key insight is to decouple hypothesis generation from prompt rewriting via a multi-agent design: a hypothesis agent proposes semantically labeled hypotheses guided by a heuristic set, while a reflection agent rewrites the prompt targeting each hypothesis independently. Parallel minibatch validation then selects the best hypothesis, constructing an interpretable optimization trace that makes every step auditable and directional. A two-layer explore-exploit mechanism (Sutton and Barto, 2018)—combining random restart for escaping seed-induced traps and epsilon-greedy sampling for hypothesis diversity—further ensures robust global search. On the same defective seed, VISTA recovers accuracy to 87.57%. Figure 2 illustrates how GEPA becomes trapped under a defective seed, whereas VISTA is able to escape.

Our contributions are as follows:

- We identify and formalize four systematic limitations of reflective APO under a unified interpretability-blindspot framework.
- We propose VISTA, a multi-agent APO framework that decouples hypothesis generation from prompt rewriting, enabling verifiable and interpretable optimization traces.
- We provide comprehensive experiments on GSM8K and AIME2025 (Balunović et al., 2025) validating both the proposed limitations and the effectiveness of VISTA.

2 Related Work

2.1 Automatic Prompt Optimization

Prompt engineering has long relied on manual design (Brown et al., 2020; Wei et al., 2022), but recent work has shifted toward automated approaches. Instruction induction methods (Honovich et al., 2023; Zhou et al., 2023) generate candidate prompts from input-output examples. Optimization-based methods treat prompt search as a black-box problem: OPRO (Yang et al., 2024) uses an LLM as a meta-optimizer guided by past scores, while APE (Zhou et al., 2023) searches over instruction candidates via sampling. Gradient-inspired methods such as ProTeGi (Pryzant et al., 2023) and TextGrad (Yuksekgonul et al., 2024) compute textual "gradients" from failure feedback to guide prompt updates. More recently, DSPy (Khattab et al., 2024) and MIPROv2 (Opsahl-Ong et al., 2024) extend APO to multi-module compound systems, jointly optimizing prompts and few-shot demonstrations.

Evolutionary approaches (Fernando et al., 2024; Guo et al., 2024) apply self-referential mutation to prompt search. GEPA (Agrawal et al., 2026) further advances this paradigm by combining reflective prompt mutation with Pareto-based candidate selection, achieving strong sample efficiency across diverse tasks. EvoX (Liu et al., 2026) proposes meta-evolution of the search strategy itself, jointly optimizing the prompt and the optimization policy.

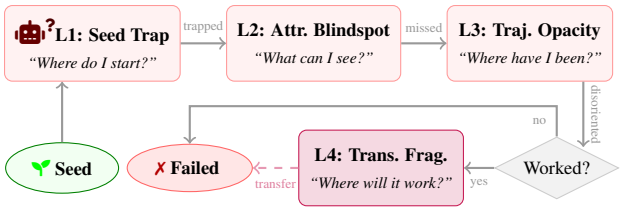


Figure 3: Four systematic limitations of reflective APO. L1–L3 form a causal chain, and L4 can apply even when optimization succeeds.

While these methods demonstrate strong empirical results, they share a common limitation: attribution generation and prompt rewriting are entangled in a single black-box step, leaving the optimization process without interpretable structure or verifiable root-cause attribution.

2.2 LLM Self-Correction and Its Limits

A prominent line of work explores iterative self-refinement of LLM outputs. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) prompt LLMs to critique and revise their own outputs across multiple rounds, demonstrating improvements on a range of tasks. However, Huang et al. (2024) provide a critical counterpoint: without access to external feedback, LLMs cannot reliably self-correct reasoning, as revisions are bounded by the same prior that produced the original error. Subsequent work has further characterized conditions under which self-correction succeeds or fails (Olausson et al., 2024; Stechly et al., 2023; Kamoi et al., 2024). These findings directly motivate VISTA: reflective APO diagnosis is subject to the same prior constraints, making external heuristics necessary.

3 Diagnosing the Black Box: Four Limitations

We now expand on each limitation introduced in Section 1. These four limitations form a progressive indictment of reflective APO: the seed constrains where search begins (L1), the reflector constrains what can be hypothesized (L2), and the absence of semantic structure constrains what the optimizer can learn from its own trajectory (L3). Even if L1–L3 were fully resolved and optimization succeeded, the result carries no guarantee of generalization across base models (L4). Figure 3 illustrates how they manifest as a causal chain.

Case: Attribution Blindspot

Optimized prompts across rounds:

Rd.1: "You are a math assistant. Solve the problem step by step. Output in JSON."

Rd.3: "You are a math assistant. Solve the problem step by step. Verify each calculation. Output in JSON."

Rd.5: "You are a math assistant. Break the problem into parts. Verify each calculation. Output in JSON."

Rd.7: "You are a math assistant. Break the problem into parts. Verify each calculation. Explain your reasoning in detail before giving the final answer. Output in JSON."

✗ **Attribution Blindspot:** All attributions target reasoning quality; the structural root cause remains outside the reflector's attribution space across all rounds and reflector configurations.

Figure 4: An illustrative case of attribution blindspot.

3.1 L1: Seed Trap

The seed prompt implicitly defines the initial search region. When the seed contains structural defects—such as incorrect output field ordering, malformed schema constraints, or contradictory instructions—the optimizer inherits these defects as implicit constraints. Because reflective APO produces no record of which seed-level assumptions are being carried forward, it has no mechanism to identify or question them.

Consider the official GEPA (Agrawal et al., 2026) seed prompt (Figure 1). Its output schema specifies `final_answer` before `solution_pad`, causing the base model to output its final answer *before* its Chain-of-Thought (CoT) reasoning, effectively preventing CoT from influencing the answer. This ordering constraint is silently inherited throughout all optimization rounds and never flagged as a candidate root cause.

3.2 L2: Attribution Blindspot

The reflector's attribution space is doubly bounded. First, structurally: the reflector can only propose root causes that fall within its prior distribution of plausible failure modes, systematically missing categories that are underrepresented or absent from that distribution. Huang et al. (2024) demonstrate that LLMs cannot reliably self-correct without external feedback, as revisions are constrained by the same prior that produced the original error—reflective APO diagnosis is subject to the same bound. Second, by capability: a weaker reflector has a narrower effective attribution space, and even

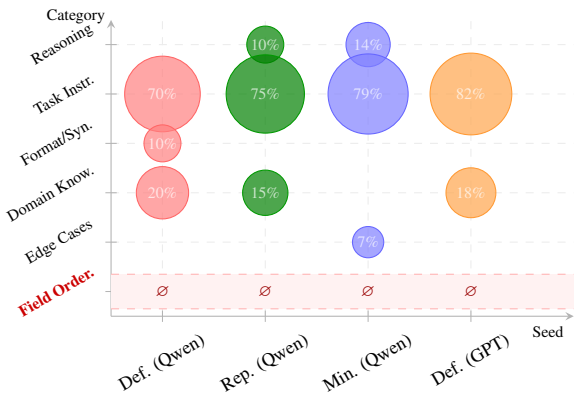
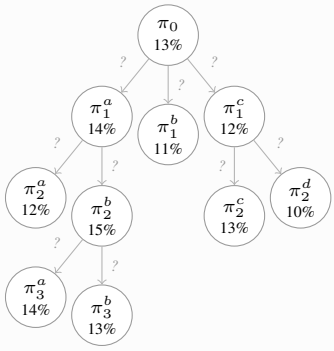


Figure 5: GEPA attribution distribution on GSM8K (Qwen3-4B base, Qwen3-8B/GPT-4o-mini reflectors).

Case: Trajectory Opacity

Optimization tree across rounds:



✗ **Trajectory opacity:** every edge carries only a question mark. The tree grows but carries no semantic meaning; the optimizer cannot detect oscillation, accumulate directional experience, or reason about its own trajectory.

Figure 6: An illustrative case of trajectory opacity.

within categories it can in principle reach, attribution quality degrades with model capability. Both constraints are invisible to the optimizer.

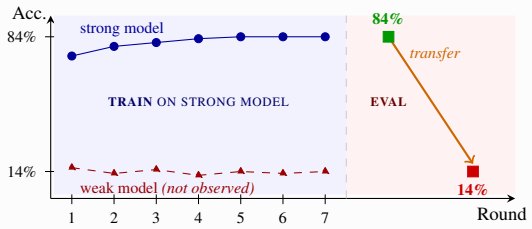
Analysis of GEPA's optimized prompts across all rounds confirms this: despite iterative optimization, no round ever corrects the field ordering defect (Figure 4). The reflector consistently attributes failures to reasoning errors, hallucinations, and instruction-following issues, never proposing structural attributions. Figure 5 shows the distribution of attribution categories across configurations—the actual root cause (`Field Ordering`) receives zero attributions across all configurations regardless of reflector strength, confirming that the blindspot is prior-structural rather than purely capability-driven.

3.3 L3: Trajectory Opacity

Even if the reflector's attribution happens to point in the right direction, the optimizer has no way

Case: Transfer Fragility

Optimization curve across rounds:



✗ **Transfer fragility:** the field order defect was never corrected because the strong model's robustness masked it. Once transferred to a weaker model, the latent defect is exposed silently.

Figure 7: An illustrative case of transfer fragility.

to know it did. GEPA does perform selection across candidate prompts, but the optimization trajectory is entirely label-free (Figure 6). Each transition from one prompt to the next is driven by an accuracy-gain signal with no record of what root cause was attributed, what change was made, or why accuracy shifted. The optimizer knows that something worked, but not what, and has no basis for reasoning about what to try next.

This semantic vacuum has two concrete consequences. First, the optimizer cannot detect oscillation: if two conflicting attributions are applied alternately across rounds, there is no signal that the same ground is being revisited. Second, the optimizer cannot accumulate directional experience: each round starts from scratch, with no memory of which attribution categories have been productive or exhausted. The optimization tree grows in size but remains semantically empty—a map with nodes and edges but no labels, making the full evolution uninterpretable.

3.4 L4: Transfer Fragility

Prompts optimized under reflective APO are implicitly tailored to the base model's behavior during optimization (Zhao et al., 2021; Lu et al., 2022). When the optimized prompt is transferred to a different base model, the structural assumptions encoded in the prompt—output format expectations, reasoning style preferences, instruction sensitivity—may no longer hold, causing silent performance degradation. Because reflective APO produces no record of which model-specific behaviors were exploited during optimization, it provides no signal about where its results will or will not generalize.

This fragility is particularly counterintuitive: a

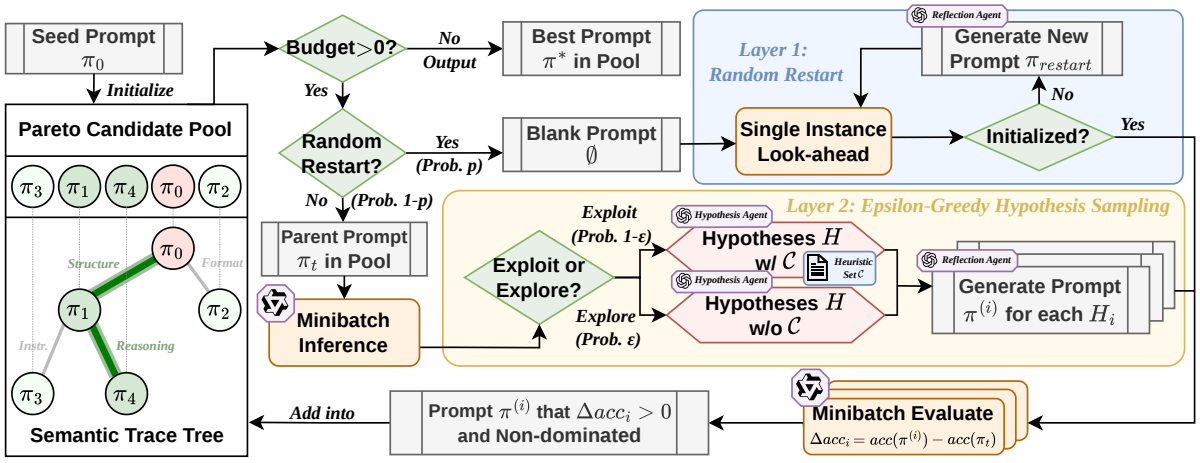


Figure 8: VISTA pipeline overview: a multi-agent framework that decouples hypothesis generation from prompt rewriting via heuristic-guided parallel verification and interpretable semantic trace trees.

prompt optimized against a stronger model may mask latent defects that the model's robustness compensates for. Once transferred to a weaker model that more faithfully follows the defective schema, the latent defect is exposed—and there is no mechanism to detect this regression without re-running evaluation on the target model (Figure 7).

4 Proposed VISTA Framework

4.1 Overview

VISTA decouples hypothesis generation from prompt rewriting via a multi-agent design (Wu et al., 2024): a *hypothesis agent* proposes semantically labeled hypotheses, and a *reflection agent* rewrites the prompt targeting each hypothesis independently. Each prompt update is thus grounded in an explicit, verifiable hypothesis, making the optimization process interpretable at every step.

The Pareto pool retains non-dominated prompts under per-sample dominance following Agrawal et al. (2026), with prompts sampled proportional to their per-sample win count. In each round, a selected prompt is passed through the two-layer explore-exploit mechanism (Section 4.4) to generate K candidate hypotheses (Section 4.2). The reflection agent produces K candidate prompts, one per hypothesis. The best-verified candidate may be added to the Pareto pool with its root-cause label, extending the semantic trace tree (Section 4.3). Figure 8 gives a visual overview, and Algorithm 1 formalizes the full procedure.

4.2 Hypothesis Generation

Each hypothesis $H_i = (c_i, d_i)$ consists of a category label $c_i \in \mathcal{C}$ and a natural language description d_i of the suspected failure mode. Let $\mathcal{F}_t$ denote the set of failure cases at round t. The label set $\mathcal{C}$ is an extensible taxonomy of heuristic failure modes curated from representative cases; each entry specifies a failure mode category, a description, and a suggested fix direction (see Appendix E).

Let $\mathcal{H}_\theta$ denote the hypothesis space of a reflector with parameters θ, and let c^* denote the true root cause. By Huang et al. (2024), $P(c^* \in \mathcal{H}_\theta) < 1$ for root causes outside the model's prior, and this probability cannot be improved by further prompting alone. An unconstrained hypothesis agent is subject to the same bound. Let $\beta = P(c^* \in \mathcal{C})$ denote the coverage probability of $\mathcal{C}$. The heuristic set introduces an external prior independent of θ, ensuring:

$$\beta > P(c^* \in \mathcal{H}_\theta) \tag{1}$$

for any c^* covered by the heuristic set.

Let π_t denote the prompt selected from $\mathcal{P}$ at round t. For each hypothesis H_i, the reflection agent independently rewrites π_t to address the hypothesized root cause, producing candidate $\pi^{(i)}$; we write $\{\pi^{(i)}\}_{i=1}^K$ for the full set of K candidates. Let $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}}$ denote the task dataset partitioned into a training split for failure cases collection and a validation split for candidate selection. Let $\mathcal{M} \sim \mathcal{D}_{\text{train}}$ denote a minibatch of size b and $\text{acc}(\pi, \mathcal{M})$ the accuracy of prompt π on $\mathcal{M}$. Define the accuracy gain of a prompt π on minibatch $\mathcal{M}$ relative to the current prompt π_t as:

$$\Delta\text{acc}(\pi, \mathcal{M}) = \text{acc}(\pi, \mathcal{M}) - \text{acc}(\pi_t, \mathcal{M}) \tag{2}$$

The accuracy gain of the i-th candidate is then $\Delta\text{acc}_i = \Delta\text{acc}(\pi^{(i)}, \mathcal{M})$, and the winning hypoth-

Algorithm 1 VISTA Prompt Optimization

Require: dataset $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}}$, seed prompt π_0, heuristic set $\mathcal{C}$, budget B, hyperparameters K, p, ε, b
Ensure: Optimized prompt π^*
  Initialize Pareto pool $\mathcal{P} \leftarrow \{\pi_0\}$, $\pi^* \leftarrow \pi_0$
  while budget $B > 0$ **do**
    $\pi_t \leftarrow \textsc{SelectPareto}(\mathcal{P})$, $\pi' \leftarrow \varnothing$
    Sample $\mathcal{M} \sim \mathcal{D}_{\text{train}}$ with $|\mathcal{M}| = b$
    if Uniform$(0,1) < p$ **then**
      $\pi_{\text{restart}} \leftarrow \textsc{Restart}(\mathcal{D}_{\text{train}})$
      if $\Delta\text{acc}(\pi_{\text{restart}}, \mathcal{M}) > 0$ **then**
        $\pi' \leftarrow \pi_{\text{restart}}$
      end if
      $B' \leftarrow b + 1$
    else
      for $i = 1, \ldots, K$ **do**
        Collect failure cases $\mathcal{F}_t$ from $\mathcal{M}$
        $H_i \leftarrow \textsc{Sample}(\mathcal{C},\ \mathcal{F}_t,\ \varepsilon)$
        $\pi^{(i)} \leftarrow \textsc{Reflect}(\pi_t,\ H_i, \mathcal{F}_t)$
      end for
      $\pi' \leftarrow \pi^{(i^*)}$, $i^* = \arg\max_{i \in \{j\,:\,\Delta\text{acc}_j > 0\}} \Delta\text{acc}_i$
      $B' \leftarrow b \cdot K$
    end if
    if $\pi' \neq \varnothing$ **then**
      Evaluate π' on $\mathcal{D}_{\text{val}}$
      if π' is not dominated by any $\pi \in \mathcal{P}$ on $\mathcal{D}_{\text{val}}$ **then**
        Add π' to $\mathcal{P}$ with label $c^* = c_{i^*}$
        $\textsc{UpdatePareto}(\mathcal{P})$
      end if
      if $\text{acc}(\pi',\ \mathcal{D}_{\text{val}}) > \text{acc}(\pi^*,\ \mathcal{D}_{\text{val}})$ **then**
        $\pi^* \leftarrow \pi'$
      end if
      $B' \leftarrow B' + |\mathcal{D}_{\text{val}}|$
    end if
    $B \leftarrow B - B'$
  end while
  return π^*

esis is selected by:

$$i^* = \arg \max_{i \in \{j\,:\,\Delta\text{acc}_j > 0\}} \Delta\text{acc}_i \tag{3}$$

The winner $\pi^{(i^*)}$ is then evaluated on $\mathcal{D}_{\text{val}}$ and added to $\mathcal{P}$ with label $c^* = c_{i^*}$ if not dominated by any $\pi \in \mathcal{P}$: the selected root cause is not what the reflector believes caused the failure, but what empirically produces the largest improvement on held-out instances.

4.3 Semantic Trace

VISTA maintains a *semantic trace tree* $\mathcal{T} = (V, E)$, where each node $v \in V$ corresponds to a prompt candidate $\pi^{(v)}$ and each directed edge $(u, v) \in E$ is annotated with the tuple (c^*, δ), where $\delta = \Delta\text{acc}$ denotes the accuracy gain of the optimization step that produced v from u. The tree is rooted at π_0 and grows by one node per successful round, giving the full optimization history a structured, auditable form: every prompt can be traced back to its root-cause chain.

The optimization trajectory up to round t is the root-to-current path:

$$\tau_t = \left(\pi_0 \xrightarrow{(c_1^*, \delta_1)} \pi_1 \xrightarrow{(c_2^*, \delta_2)} \cdots \xrightarrow{(c_t^*, \delta_t)} \pi_t \right) \tag{4}$$

where $c_k^* \in \mathcal{C}$ is the selected root-cause label at round k. Unlike GEPA's unlabeled sequence, τ_t is fully interpretable: each transition has a causal explanation and a verified performance delta $\Delta\text{acc}^{(t)}$.

The trace τ_t is provided to the hypothesis agent as context at each round, enabling it to avoid already-explored directions, identify diminishing-return categories, and detect when two labels alternate without joint resolution. In the latter case, the agent generates a joint hypothesis addressing both dimensions simultaneously, collapsing what would otherwise require multiple sequential rounds into a single update.

4.4 Two-Layer Explore-Exploit

VISTA introduces a two-layer explore-exploit mechanism (Sutton and Barto, 2018) targeting L1 and L2 respectively.

Layer 1: Random Restart. At the start of each round, with probability $p \in (0, 1)$, VISTA triggers a random restart (Lourenço et al., 2003). A *blank prompt look-ahead* is executed iteratively: a single training instance is run under the current prompt, collecting the raw output o_{raw} and any parsing error e. The reflection agent constructs a new prompt conditioned on (o_{raw}, e), which is then used in the next look-ahead step. This loop repeats until $e = \varnothing$, at which point π_{restart} is considered initialized from the model's natural behavior rather than from any inherited seed constraints. Formally, while standard mutation conditions on π_t:

$$\pi^{(i)} = f_{\text{reflect}}(\pi_t, H_i, \mathcal{F}_t) \tag{5}$$

random restart conditions on the model's natural behavior, using a fixed null hypothesis $H_{\text{blank}} = (\text{none},\ \text{“initialize from model output”})$ that carries no prior attribution:

$$\pi_{\text{restart}} = f_{\text{reflect}}(\emptyset,\ H_{\text{blank}},\ o_{\text{raw}}) \tag{6}$$

This ensures the restarted prompt reflects what the model naturally produces, not what the seed constrains it to produce. The resulting π_{restart} is then evaluated against π_t on $\mathcal{M}$; it is added to $\mathcal{P}$ only if $\Delta\text{acc}(\pi_{\text{restart}}, \mathcal{M}) > 0$ and not dominated by any $\pi \in \mathcal{P}$ on $\mathcal{D}_{val}$. The cost of a restart is a few look-ahead steps, making it negligible relative to the minibatch validation budget.

Table 1: Accuracy (%) across benchmarks, conditions, and methods.

	GSM8K			AIME2025		
Method	**Defective**	**Repaired**	**Minimal**	**Defective**	**Repaired**	**Minimal**
No Opt.	23.81	85.59	20.67	38.67	40.00	40.00
GEPA	13.50	86.53	21.68	44.00	39.33	42.00
VISTA	**87.57**	**87.34**	**85.67**	**46.00**	**46.67**	**44.00**

Layer 2: Epsilon-Greedy Hypothesis Sampling. Within each round, the K hypotheses are drawn according to an epsilon-greedy strategy. For each of the K slots:

$$H_i \sim \begin{cases} \text{Heuristic}(\mathcal{C},\ \mathcal{F}_t) & \text{with probability } 1-\varepsilon \\ \text{Free}(\mathcal{F}_t) & \text{with probability } \varepsilon \end{cases} \quad (7)$$

The heuristic set branch exploits known failure mode categories; the free branch generates unconstrained hypotheses, allowing discovery of failure modes outside $\mathcal{C}$. In expectation, $\lfloor (1-\varepsilon)K \rfloor$ hypotheses target known modes and $\lceil \varepsilon K \rceil$ explore novel ones.

Sample Efficiency. For the purposes of this analysis, we treat β as VISTA's effective per-round probability of identifying c^*, lower-bounded by $P(c^* \in \mathcal{C})$ as defined in Section 4.2. Let $\alpha \in [0, 1)$ denote GEPA's empirical probability of selecting c^* in any given round. The analysis in Section 3.2 suggests that α is empirically close to zero for structural failure modes, while $\beta > \alpha$ follows from $\mathcal{C}$ explicitly covering structural categories.

Assuming that each round independently samples a candidate root cause with fixed probability (approximated as geometric trials), the expected number of rounds until c^* is first identified satisfies

$$\mathbb{E}[N_{\text{VISTA}}] \leq \frac{1}{\beta} < \frac{1}{\alpha} = \mathbb{E}[N_{\text{GEPA}}], \quad \alpha > 0 \quad (8)$$

where N denotes the number of rounds until c^* is first selected as i^*. The inequality follows from $\beta > \alpha > 0$. In the limiting case where $\alpha \to 0$, $\mathbb{E}[N_{\text{GEPA}}]$ diverges, which is consistent with the empirical observation that GEPA fails to identify the structural root cause across all rounds.

5 Experiments

5.1 Experimental Setup

Models and Benchmarks. For GSM8K (Cobbe et al., 2021) experiments, we use Qwen3-4B as the base model and Qwen3-8B (Yang et al., 2025) as the reflector. For AIME2025 (Balunović et al., 2025) experiments, we use GPT-4.1-mini (OpenAI, 2025) as the base model and GPT-4o-mini (OpenAI, 2024) as the reflector.

Conditions. We evaluate under three seed conditions to stress-test optimizer robustness: (1) Defective seed: the official GEPA(Agrawal et al., 2026) seed prompt, which contains an inverted output field order that silently disables CoT reasoning; (2) Repaired seed: a manually corrected version with the field order fixed; (3) Minimal seed: a single-sentence prompt with no structural constraints.

Baselines. We compare against two baselines: No optimization, which evaluates the seed prompt directly, and GEPA, the state-of-the-art reflective APO method.

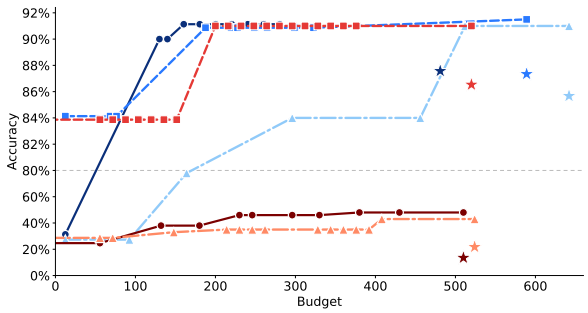
VISTA Configuration. Unless otherwise specified, VISTA uses $K = 3$ hypotheses per round, restart probability $p = 0.2$, and exploration rate $\varepsilon = 0.1$. (see Appendix B for full setup details).

5.2 Main Results

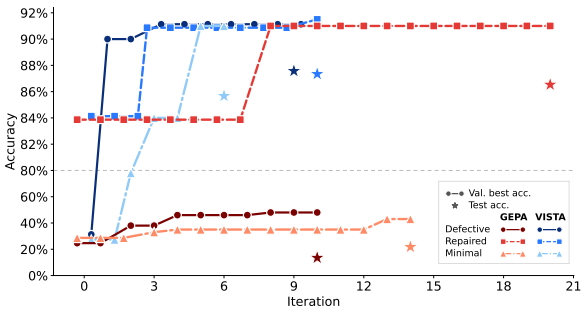
Table 1 reports results across all seed conditions, methods, and benchmarks.

GSM8K. Under the defective seed, GEPA degrades accuracy from 23.81% to 13.50% while VISTA recovers to 87.57% (+74.07 points), directly validating L1 and providing evidence for L2. The root cause—inverted output field order—is never hypothesized by GEPA across all optimization rounds even with full failure information, confirming that the hypothesis space is the binding constraint rather than information availability. As shown in Figure 9b, GEPA's best-score trajectory remains flat throughout optimization while VISTA converges within the first few rounds, and this advantage holds under the same budget (Figure 9a).

Under the repaired seed, all methods converge to similar accuracy, confirming VISTA introduces no regression on well-formed seeds. Under the minimal seed, GEPA again fails to escape the low-accuracy region (20.67% → 21.68%), while VISTA



(a) Accuracy (%) vs. metric calls.



(b) Accuracy (%) vs. optimization rounds.

Figure 9: Optimization curves on GSM8K under the defective seed (best score to date).

Table 2: GSM8K accuracy (%) on the defective seed. Columns 1–2: Qwen3-4B base, Qwen3-8B/GPT-4o-mini reflectors. Column 3: trained on GPT-4.1-mini (GPT-4o-mini reflector), evaluated on Qwen3-4B.

	Reflector		
Method	**Qwen3-8B**	**GPT-4o-mini**	**Cross-Model**
GEPA	13.50	23.43	22.74
VISTA	**87.57**	**87.64**	**86.05**

recovers to 85.67%, demonstrating that VISTA's gains are not specific to the field-ordering defect.

Table 2 validates L2 and L4. Across both reflector configurations, GEPA remains near baseline while VISTA maintains strong performance, confirming the blindspot is prior-structural rather than capability-driven. For L4, GEPA's cross-model accuracy (22.74%) remains near its single-model baseline (13.50%), confirming that optimization gains under GEPA do not transfer across base models. By contrast, VISTA maintains strong performance under transfer (86.05%), suggesting that heuristic-guided optimization produces more structurally generalizable prompts.

AIME2025. Absolute gains are smaller than on GSM8K, reflecting the task's lower sensitivity to prompt structure under high problem difficulty. Nevertheless, VISTA consistently outperforms GEPA across all seed conditions, and GEPA degrades below the no-optimization baseline under the repaired seed (39.33% vs. 40.00%), further confirming the instability of unconstrained reflection.

5.3 Ablation Study

We conduct ablation experiments on GSM8K under the defective seed. All ablations fix Qwen3-8B as the reflector. Table 3 reports all results.

Two findings stand out. First, heuristic guidance is the dominant factor: removing exploitation

Table 3: Ablation results on GSM8K (%) under the defective seed (Qwen3-4B base, Qwen3-8B reflector).

	Hyper.			
Configuration	K	p	ε	**Acc.**
Effect of K				
VISTA	3	0.2	0.1	87.57
VISTA, K=1	1	0	0.1	75.97
VISTA, K=3	3	0	0.1	86.81
VISTA, K=5	5	0	0.1	83.89
Effect of Heuristic Set				
VISTA	3	0.2	0.1	87.57
w/o Exploration	3	0	0	85.60
w/o Exploitation	3	0	1.0	22.97
Component Contribution				
GEPA	–	–	–	13.50
+ Restart	0	0.2	0	15.69
+ Parallel Sampling	3	0.2	0	20.17
+ Heur.-Guided Reflection	3	0.2	0	79.98

($\varepsilon = 1.0$) collapses accuracy to 22.97%, while removing exploration ($\varepsilon = 0$) causes only a modest drop to 85.60%. Second, the component contribution rows confirm that the heuristic set accounts for the largest single gain (+59.81 points), with random restart and parallel sampling contributing cumulatively but secondarily. $K = 3$ provides the best tradeoff; performance degrades at $K = 5$, suggesting noise from additional candidates outweighs their diversity benefit.

6 Conclusion

We identified four systematic limitations of reflective APO methods under a unified interpretability-blindspot framework, and demonstrated them concretely: on GSM8K with a defective seed, GEPA degrades accuracy from 23.81% to 13.50% while the root cause is never identified across all optimization rounds. We proposed VISTA, a multi-agent APO framework that decouples hypothesis generation from prompt rewriting, replacing black-box

reflection with heuristic-guided parallel verification and interpretable semantic trace trees. VISTA recovers accuracy to 87.57% on the same defective seed and maintains strong performance across all configurations. We hope this work motivates a broader shift toward interpretable, robust optimization in the APO paradigm.

Limitations

VISTA directly addresses L1, L2, and L3, and partially mitigates L4: heuristic-guided optimization produces structurally grounded prompts that transfer more reliably across base models than unconstrained reflection, as evidenced by Table 2. Transfer fragility nonetheless remains open in general—VISTA provides no explicit signal about generalization, and its transfer advantage may not hold across model families with larger capability gaps. Incorporating multi-model minibatch validation, where hypothesis selection is based on accuracy gains averaged across a distribution of base models, is a natural extension. Beyond L4, several further limitations apply.

Heuristic coverage. The heuristic set has finite coverage; failure modes outside its scope can only be discovered via the ε-exploration branch, which has lower expected sample efficiency than heuristic-guided search. This motivates more principled exploration strategies.

Hyperparameter sensitivity. VISTA introduces three hyperparameters (K, p, ε) that interact with the rollout budget in non-obvious ways. Our ablations identify $K{=}3$, $p{=}0.2$, $\varepsilon{=}0.1$ as a robust default, but optimal configurations likely vary across tasks and budget constraints.

Reasoning models. Our analysis focuses on standard instruction-following models. Modern reasoning models generate explicit chain-of-thought traces before producing output, which may partially circumvent the field-ordering defect in L1 regardless of output schema order. The seed trap and attribution blindspot may therefore be less severe for such models, and the structural gains from VISTA's heuristics could diminish accordingly.

Benchmark coverage. Experiments are confined to mathematical reasoning benchmarks (GSM8K and AIME2025). Whether the four limitations and VISTA's mitigations generalize to more open-ended domains—such as long-form generation, dialogue, or coding—remains an open question, as these domains may exhibit different failure mode distributions that the current heuristic set does not cover.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and constructive suggestions, which helped improve the quality of this paper. The first author is also grateful to the University of California, Berkeley, for providing the valuable opportunity for an academic visit. Furthermore, we extend our sincere gratitude to the professors there for the precious opportunities for academic exchange and insightful discussions.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. Matharena: Evaluating llms on uncontaminated math competitions.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2024. Instructzero: efficient instruction optimization for black-box large language models6518. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: self-referential

self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*.

Or Honovich, Uri Shaham, Samuel R. Bowman, and Omer Levy. 2023. Instruction induction: From few examples to natural language task descriptions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1935–1952, Toronto, Canada. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

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. Curran Associates, Inc.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, and 1 others. 2026. Evox: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*.

Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. 2003. *Iterated Local Search*, pages 320–353. Springer US, Boston, MA.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, Dublin, Ireland. 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 *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2024. Gpt-4o mini: advancing cost-efficient intelligence.

OpenAI. 2025. Introducing GPT-4.1 in the API.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Herbert Robbins. 1952. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv e-prints*, pages arXiv–2402.

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, pages 8634–8652. Curran Associates, Inc.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. 2023. GPT-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Richard Sutton and Andrew Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed 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*, volume 35, pages 24824–24837. Curran Associates, Inc.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Qinyuan Ye, Mohamed Ahmed, Reid Pryzant, and Fereshte Khani. 2024. Prompt engineering a prompt engineer. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 355–385, Bangkok, Thailand. Association for Computational Linguistics.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *Preprint*, arXiv:2406.07496.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A Discussion

A.1 Data-Driven Heuristics

The current heuristic set is manually curated from representative failure cases and modes. While this ensures coverage of known out-of-prior failure modes, the approach does not scale and may miss task-specific failure patterns underrepresented in the literature.

A more scalable alternative is data-driven construction: collecting large-scale optimization trajectories across diverse tasks and reflectors, clustering failure cases by semantic similarity, and distilling recurring patterns into heuristic categories automatically. Frequently-winning free hypotheses from the ε-exploration branch are a natural seed for this process—each successful free hypothesis that recurs across multiple runs is a candidate for promotion, enabling the heuristic set to grow with accumulated experience.

This process has a bootstrap dependency, however: early trajectories are collected under the manually curated heuristic set and may underrepresent failure modes outside its scope. Seeding with diverse random-restart trajectories, where $p = 1$ throughout, provides a less biased initialization for the data-driven construction process.

A.2 Broader Semantic Trace Utility

The semantic trace tree already enables interpretable optimization histories and oscillation detection, but its potential extends further. The trace encodes a causal graph of which hypotheses led to which improvements, providing rich information that could actively guide future decisions—for instance, informing the System-Aware Merge (Agrawal et al., 2026) step by preferring merges between candidates whose root-cause label sequences are complementary, or prioritizing hypothesis categories that have historically been productive. We view the semantic trace as a general-purpose interpretability substrate for APO, and expect its utility to grow as more sophisticated uses are developed.

The semantic trace also has potential for cross-task transfer: hypothesis categories that consistently produce large δ gains across multiple tasks constitute a task-agnostic prior that can warm-start optimization on new tasks, reducing the rounds needed to identify the dominant failure mode.

A.3 Adaptive Explore-Exploit Scheduling

The current two-layer mechanism uses fixed p and ε throughout optimization. In practice, the optimal balance shifts as optimization progresses: early rounds benefit from broader exploration to identify the dominant failure mode, while later rounds benefit from focused exploitation once a productive direction has been found.

The semantic trace provides a natural signal for adaptive scheduling. When the trace shows consistent improvement in a single hypothesis category, ε could be reduced to concentrate resources on exploitation. When oscillation is detected or improvement plateaus, p could be temporarily increased to trigger more aggressive restart behavior. Formalizing this as a bandit problem (Robbins, 1952) over the heuristic set—where the reward signal for each category $c \in \mathcal{C}$ is its empirical mean Δacc across historical selections—is a principled direction for future work.

B Experimental Setup (Contd.)

Hardware. All experiments were conducted on a single server equipped with one NVIDIA RTX 4090 (24 GB). Local inference for Qwen3-4B and Qwen3-8B was served via Ollama; GPT-4.1-mini and GPT-4o-mini were accessed through the OpenAI API.

Experimental Parameters. All experiments share the following configuration: minibatch size $b = 8$, budget $T = 500$, training/validation sizes of 50 each, maximum parallelism of 4 workers, and random seed 0. Evaluation uses exact-match accuracy. For GEPA, we use the default hyperparameters from the original implementation. For VISTA, the default configuration is $K = 3$, $p = 0.2$, and $\varepsilon = 0.1$; ablation groups vary one parameter at a time.

Datasets. For GSM8K, we use the official `openai/gsm8k` dataset; working training and validation sets are both sampled from the official training split and fixed for each run. The full official test split (1,319 examples) serves as the test set. For AIME, working training and validation sets are sampled from `AI-MO/aimo-validation-aime` (problems from AIME 2022–2024) and fixed for each run. The test set consists of the 30 problems from `MathArena/aime_2025` (AIME 2025 I & II), each repeated 5 times to reduce evaluation variance, yielding 150 test instances in total.

Model Parameters. All local models are loaded in bfloat16 precision without quantization. The base Qwen-3 model uses `temperature`$=0.6$, `top-p`$=0.95$, `top-k`$=20$, `min-p`$=0$, and `presence_penalty`$=1.5$, with thinking mode disabled (`reasoning_effort=none`). Hypothesis and reflection agents use default sampling parameters. Maximum generation length is set to the model default for all roles.

C Computational Cost

All local model inference (Qwen3-4B, Qwen3-8B) incurs no API cost. For experiments involving OpenAI models, total expenditure across all reported groups amounts to approximately $34.82–51.92. On GSM8K with a local base model (Qwen3-4B) and GPT-4o-mini reflector, a single VISTA run ($T=500$, $K=3$) costs $0.20 and a single GEPA run costs $0.12. On AIME2025 with GPT-4.1-mini as base model, a VISTA run costs $4.1–6.0 and a GEPA run costs $3.7–5.6. The marginal cost of the hypothesis agent relative to GEPA is $0.08–0.40 per run, confirming that VISTA's diagnostic capability comes at negligible additional expense.

D Optimization Trees

Figures 10 and 11 show the optimization trees for GEPA and VISTA under the defective seed (Qwen3-4B base, Qwen3-8B reflector, GSM8K), where each node $v \in V$ corresponds to a prompt candidate and each edge $(u, v) \in E$ records the root-cause label and accuracy gain δ of the optimization step that produced v from u. We denote the initial base prompt as π_0. For all subsequent nodes, we denote accepted prompts as $\pi_i^{(k)}$ and rejected candidates as $r_i^{(k)}$, where the subscript i is the iteration index in which the candidate was generated, and the superscript k is its candidate index within that iteration. GEPA's tree carries no semantic labels on any edge; VISTA's tree annotates every transition with (c^*, δ). Red and blue edges indicate accepted updates within an iteration, for GEPA and VISTA, respectively.

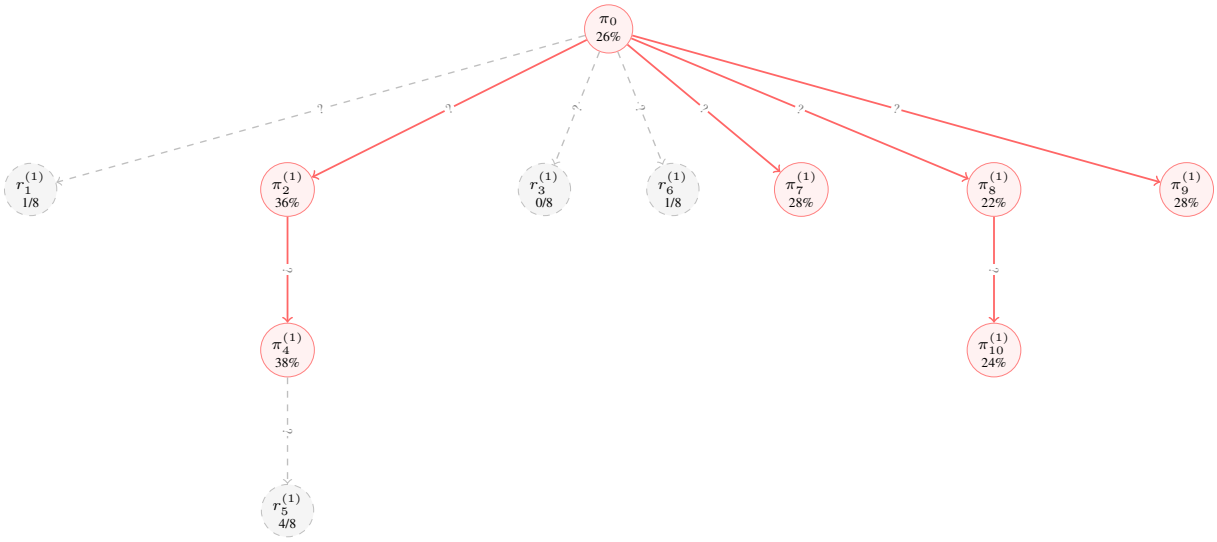


Figure 10: GEPA optimization tree under the defective seed. Every edge carries only a question mark; no root-cause label or accuracy gain δ is recorded for any transition. The structural root cause is never identified and optimization stagnates at 38%.

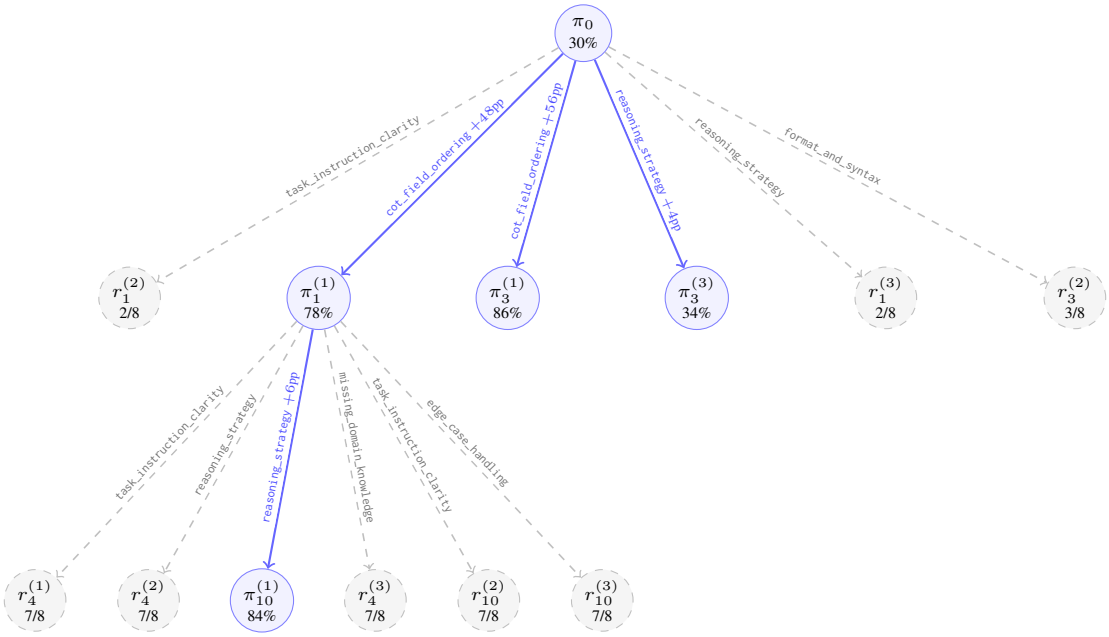


Figure 11: VISTA optimization tree under the defective seed. Every edge indicates a hypothesis annotated with a root-cause label and δ. Iteration 1 immediately identifies `cot_field_ordering` as the structural root cause, achieving a $+48$pp jump to 78% accuracy; iteration 3 further reaches 86% via the same diagnosis.

Figures 12 and 13 show the optimization trees for GEPA and VISTA under the repaired seed, following the same experimental setting and notation as above.

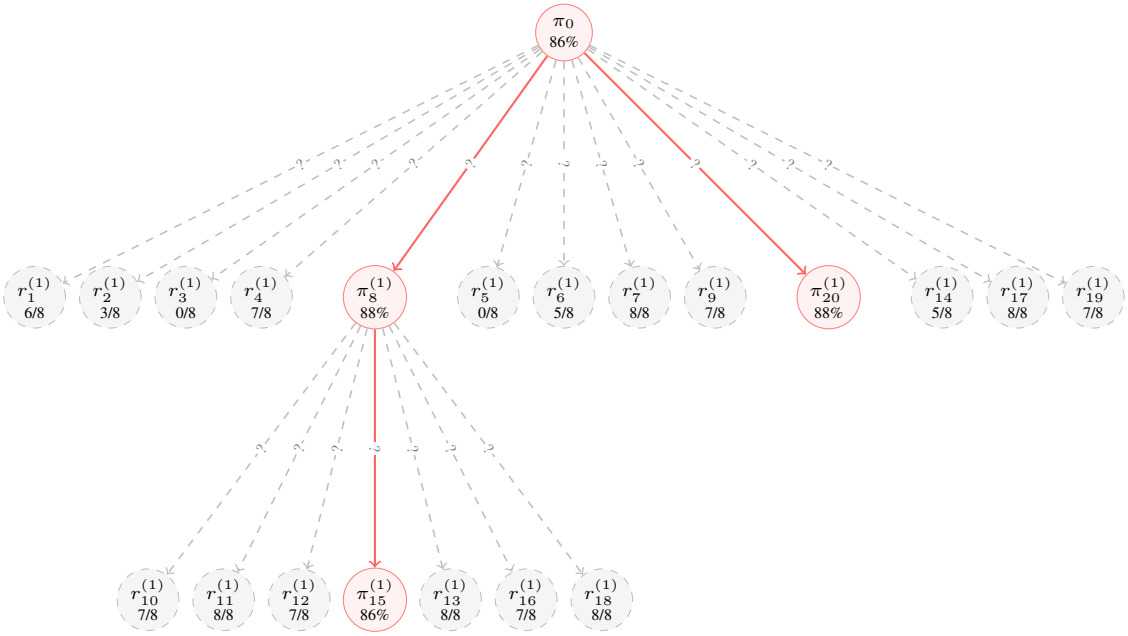


Figure 12: GEPA optimization tree under the repaired seed. The trace follows a single-candidate mutation path; two independent branches successfully reach 88% accuracy, while one subsequent update on the left branch regresses to 86%.

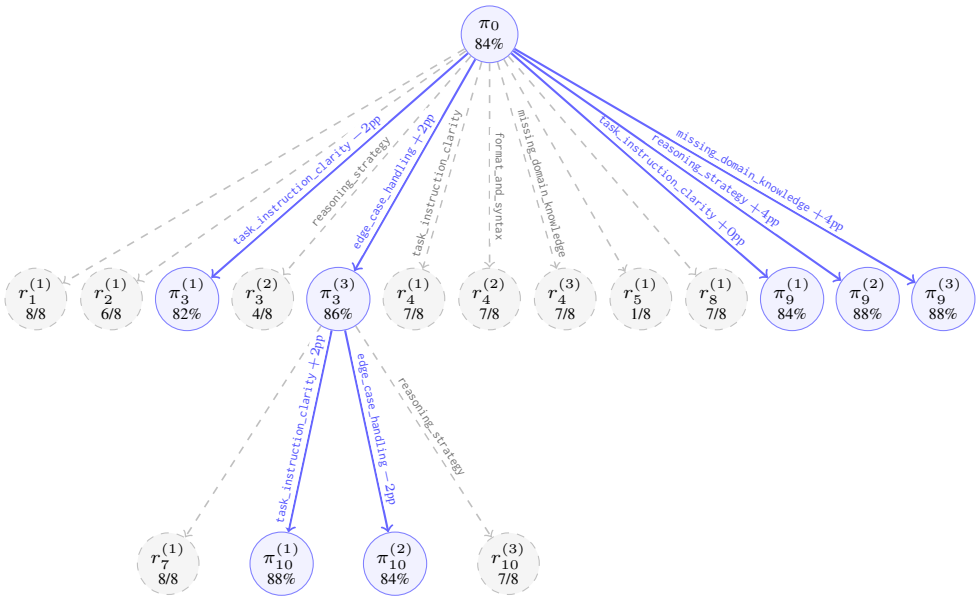


Figure 13: VISTA optimization tree under the repaired seed. The run branches early through parallel hypotheses in iterations 3, 4, and 9, reaching 88% peak accuracy via multiple distinct, semantically tagged paths. Unlabeled edges indicate iterations where no failure cases were collected on the minibatch of its parent, falling back to a single-mutation step as in GEPA.

Figures 14 and 15 show the optimization trees for GEPA and VISTA under the minimal seed, following the same experimental setting and notation as above.

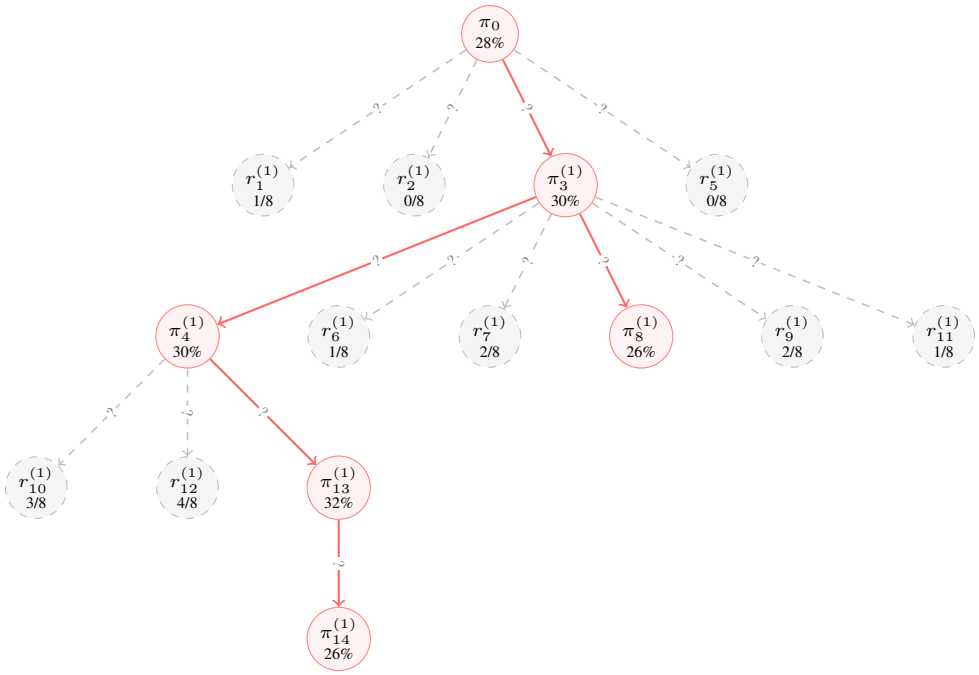


Figure 14: GEPA optimization tree under the minimal seed. The main chain reaches a peak of 32% at iteration 13 before regressing to 26% in the next update.

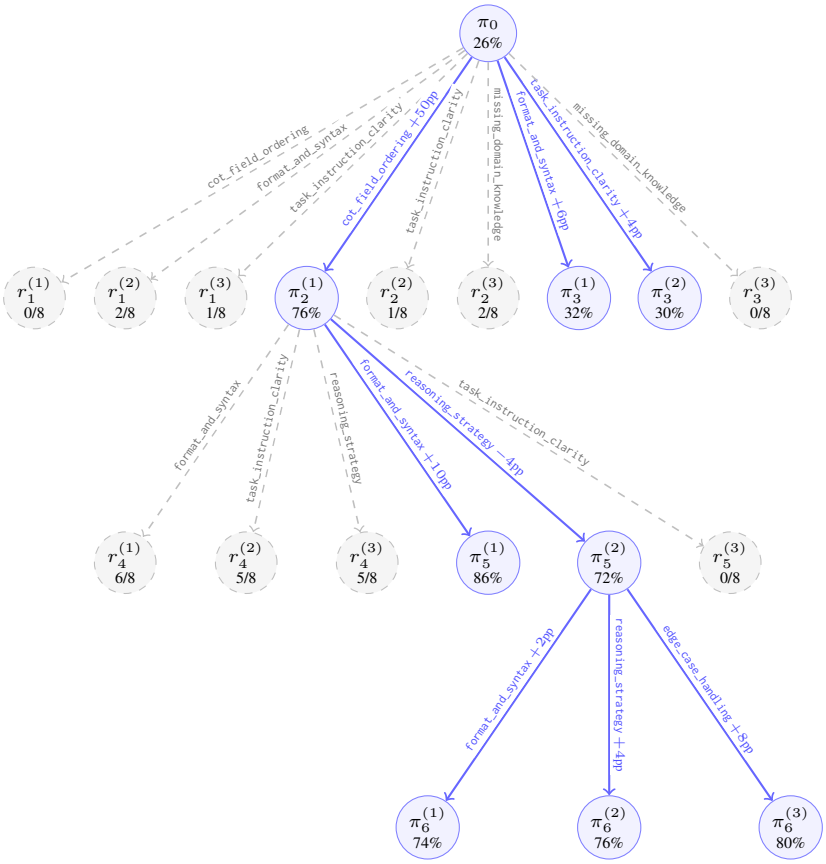


Figure 15: VISTA optimization tree under the minimal seed. Iteration 2 immediately identifies `cot_field_ordering` as a high-impact root cause (+50pp), then improves further through semantically tagged branches.

E VISTA Prompt Templates

VISTA uses two prompt templates: the hypothesis prompt instructs the hypothesis agent to generate semantically labeled root-cause hypotheses guided by the heuristic set; the reflection prompt instructs the reflection agent to rewrite the seed prompt targeting a specific hypothesis.

E.1 Hypothesis Agent Prompt

Hypothesis Agent Prompt

```
You are an expert prompt engineer analyzing why a system prompt causes failures on certain inputs.

CURRENT SYSTEM PROMPT:
{curr_instructions}

ERROR TAXONOMY:
{error_taxonomy}

FAILED SAMPLES:
{failed_samples}

TASK: Analyze the failed samples and generate exactly {num_hypotheses} diverse root-cause hypotheses.

For EACH hypothesis:
1. Select the most fitting category from the Error Taxonomy above (use the exact id field).
2. Provide a concise description of the specific root cause you identified.
3. Suggest a concrete fix direction for the prompt.

IMPORTANT:
- Each hypothesis MUST address a DIFFERENT aspect of the failures.
- Try to cover as many different taxonomy categories as possible.
- Be specific about what exactly in the current prompt causes the failure.

You MUST respond using EXACTLY the format below. Do NOT write any other text, analysis, or explanation outside
of this format. Do NOT use markdown headers or bullet points. Just output exactly {num_hypotheses} blocks in
this format:

[HYPOTHESIS 1]
TAG: {taxonomy_id}
DESCRIPTION: <one or two sentences describing the specific root cause>
FIX: <one or two sentences describing how to fix the prompt>

[HYPOTHESIS 2]
TAG: {taxonomy_id}
DESCRIPTION: <one or two sentences describing the specific root cause>
FIX: <one or two sentences describing how to fix the prompt>

[HYPOTHESIS 3]
TAG: {taxonomy_id}
DESCRIPTION: <one or two sentences describing the specific root cause>
FIX: <one or two sentences describing how to fix the prompt>

Start your response immediately. Do not include any preamble.
```

Heuristic Set Prompt

```
- id: cot_field_ordering
 name: CoT / Output Field Ordering Defect
 description: The output schema requires the final answer before the reasoning steps, preventing chain-of-thought
 from influencing the result.
- id: format_and_syntax
 name: Format / Syntax Defect
 description: The prompt does not strictly enforce output schema, key set, or syntax validity.
- id: task_instruction_clarity
 name: Task Instruction / Constraint Defect
 description: Task goals or constraints are ambiguous, contradictory, or incomplete.
- id: reasoning_strategy
 name: Reasoning Strategy / Logic Defect
 description: The prompt implies a flawed or suboptimal reasoning procedure for the task.
- id: missing_domain_knowledge
 name: Missing Domain Knowledge Gap
 description: The prompt lacks necessary domain facts or definitions required for solving.
- id: edge_case_handling
 name: Edge Case / Boundary Defect
```

```
  description: The prompt handles common inputs but fails on boundary or atypical cases.
- id: unclassified_custom
  name: Unclassified / Custom Discovery
  description: None of the predefined categories fit; discover and justify a latent failure mode.
```

E.2 Reflection Agent Prompt

Reflection Agent Prompt

```
You are a prompt optimization expert. Given a prompt, a diagnosed root cause, and a set of failure cases, your
task is to rewrite the prompt to fix the identified issue.

Root cause label: {label}
Hypothesis: {hypothesis}
Suggested fix: {suggestion}

Current prompt:
{prompt}

Failure cases:
{failure_cases}

Rewrite the prompt to address the identified root cause. Follow these rules:

- Make targeted edits only. Do not change parts of the prompt unrelated to the root cause.
- Preserve the output schema and JSON format unless the root cause is structure.
- Output only the rewritten prompt, with no explanation or preamble.
```

F Seed Prompts

All experiments use one of three seed prompts. The defective seed is the official GEPA seed for GSM8K, which contains the field order defect described in Section 3.1. The repaired seed corrects this defect manually. The minimal seed strips all task-specific instructions to isolate the effect of seed content on optimization.

F.1 Defective Seed

Defective Seed

```
You are an AI assistant that solves mathematical word problems. You will be given a question and you need to
provide a step-by-step solution to the problem. Finally, you will provide the answer to the question. When
outputting the final answer, make sure there are no other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "final_answer": <the final answer to the question> ,
  "solution_pad": <the step-by-step solution to the problem>
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any
deviation from this format will result in an error.
```

✗ **Field order defect:** `final_answer` precedes `solution_pad`, causing CoT to be generated after the answer.

F.2 Repaired Seed

Repaired Seed

```
You are an AI assistant that solves mathematical word problems. You will be given a question and you need to
provide a step-by-step solution to the problem. Finally, you will provide the answer to the question. When
outputting the final answer, make sure there are no other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "solution_pad": <the step-by-step solution to the problem>,
  "final_answer": <the final answer to the question>
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any
deviation from this format will result in an error.
```

✔ **Field order corrected:** `solution_pad` precedes `final_answer`.

F.3 Minimal Seed

Minimal Seed

```
Solve and output in a single json:
{
  "final_answer": <answer>
}
```

⚠ **Minimal instructions:** no task-specific guidance, minimal format constraint.

G Optimization Traces

We present optimization traces for GEPA and VISTA under all three seed conditions—defective, repaired, and minimal—using the Qwen3-4B base model and the Qwen3-8B reflector on GSM8K. Yellow highlighting marks changes introduced relative to the parent prompt.

G.1 Defective Seed

G.1.1 GEPA

Despite iterative optimization, the field order defect (`final_answer` before `solution_pad`) is preserved in every round. All modifications target reasoning quality.

Defective Seed, GEPA, Iteration 1 — **Parent: Seed**

```
New subsample score 1.0 is not better than old score 2.0, skipping
```

Defective Seed, GEPA, Iteration 2 — **Parent: Seed**

```
You are an AI assistant that solves mathematical word problems.
You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no
other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "final_answer": <the final answer to the question>,
  "solution_pad": <the step-by-step solution to the problem>
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation
from this format will result in an error.

Key guidelines to avoid errors:

1. **Parse problem terms precisely**:
  * "Round trip" means both up and down (double the one-way distance).
  * "More than half" implies adding to half of a value (e.g., "6 more than half of X" = (X/2) + 6).
  * "Discounts" apply sequentially (e.g., 20% off followed by 25% off the discounted price).
  * "Tips" are calculated as a percentage of the original cost, not the discounted price.

2. **Use exact arithmetic**:
  * Verify all calculations (e.g., 250 + 375 + 320 = 945, not 940).
  * Ensure multiplication and addition steps are correct (e.g., 945 × 2 = 1890, not 1980).

3. **Include all steps in the solution_pad**:
  * Show intermediate results explicitly (e.g., "Ella's score = 40 - 4 = 36").
  * Clarify reductions (e.g., "Marion's score = (36/2) + 6 = 24").
  * Account for all conditions (e.g., "Pups eaten = 56 × 2 = 112; surviving pups = 336 - 112 = 224").

4. **Validate the final answer**:
  * Ensure the answer matches the problem's requirements (e.g., "Total mice = adults + surviving pups = 56 +
224 = 280").
  * Avoid rounding errors or misinterpretations of percentages (e.g., 20% of $15 = $3, not $2.50).

5. **Format the JSON strictly**:
  * Do not include any text or explanations outside the JSON object.
  * Ensure the final answer is a number (e.g., `280`, not `"280"`).
```

Defective Seed, GEPA, Iteration 3 — **Parent: Seed**

```
New subsample score 0.0 is not better than old score 3.0, skipping
```

Defective Seed, GEPA, Iteration 4 — **Parent: Iteration 2**

```
You are an AI assistant that solves mathematical word problems.
You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no
other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "final_answer": <the final answer to the question>,
  "solution_pad": <the step-by-step solution to the problem>
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation
from this format will result in an error.

Key guidelines to avoid errors:
1. **Parse problem terms precisely**:
  * "Round trip" means both up and down (double the one-way distance).
  * "More than half" implies adding to half of a value (e.g., "6 more than half of X" = (X/2) + 6).
  * "Discounts" apply sequentially (e.g., 20% off followed by 25% off the discounted price).
  * "Tips" are calculated as a percentage of the original cost, not the discounted price.
  * "Time differences" require calculating arrival vs. departure times (e.g., "missed the bus by X minutes"
means arrival time - departure time).
2. **Use exact arithmetic**:
  * Verify all calculations (e.g., 250 + 375 + 320 = 945, not 940).
  * Ensure multiplication and addition steps are correct (e.g., 945 × 2 = 1890, not 1980).
  * Avoid rounding errors (e.g., 20% of $15 = $3, not $2.50).
  * For percentage reductions, calculate based on the original value (e.g., "1/10 less" = original - (original
× 1/10)).
3. **Include all steps in the solution_pad**:
  * Show intermediate results explicitly (e.g., "Ella's score = 40 - 4 = 36").
  * Clarify reductions (e.g., "Marion's score = (36/2) + 6 = 24").
  * Account for all conditions (e.g., "Pups eaten = 56 adults × 2 = 112; surviving pups = 336 - 112 = 224").
  * For time-based problems, calculate arrival vs. departure times explicitly (e.g., "Arrived at 8:20, bus left
at 8:00 → missed by 20 minutes").
4. **Validate the final answer**:
  * Ensure the answer matches the problem's requirements (e.g., "Total mice = adults + surviving pups = 56 +
224 = 280").
  * Avoid misinterpretations of percentages or fractions (e.g., "1/10 less" = original × 9/10).
5. **Format the JSON strictly**:
  * Do not include any text or explanations outside the JSON object.
  * Ensure the final answer is a number (e.g., `280`, not `"280"`).
  * Use only valid JSON syntax (e.g., avoid expressions like `20 + 18 + 54` in the final answer; compute the
value instead).
```

Defective Seed, GEPA, Iteration 5 — **Parent: Iteration 4**

```
New subsample score 4.0 is not better than old score 4.0, skipping
```

Defective Seed, GEPA, Iteration 6 — **Parent: Seed**

```
New subsample score 1.0 is not better than old score 1.0, skipping
```

Defective Seed, GEPA, Iteration 7 — **Parent: Seed**

```
You are an AI assistant that solves mathematical word problems.
You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no
other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "final_answer": <the final answer to the question>,
  "solution_pad": <the step-by-step solution to the problem>
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation
from this format will result in an error.

**Key Requirements:**
```

```
1. **Problem Breakdown:**
  * Identify the key variables and relationships in the problem.
  * Use precise mathematical operations (e.g., fractions, percentages, arithmetic progression).
  * Ensure calculations align with the problem's wording (e.g., "100 more than half as many" requires halving first, then adding).
2. **Solution Steps:**
  * Clearly outline each step in plain text, avoiding markdown.
  * Include intermediate calculations (e.g., "Half of 3000 is 1500").
  * Verify that all operations are logically derived from the problem's constraints.
3. **Final Answer:**
  * Provide the exact numerical answer (e.g., "2600") without text or units.
  * Ensure the answer matches the correct calculation, as highlighted in feedback.
4. **Edge Cases:**
  * Handle unit conversions (e.g., miles, kg) and spatial constraints (e.g., spacing between objects).
  * Account for rounding rules (e.g., integer results for physical quantities).
5. **Validation:**
  * Cross-check steps to avoid errors (e.g., in Example 7, ensure subtraction of miles is correct).
  * Use the problem's context to validate feasibility (e.g., maximum weight capacity in Example 6).
```

Defective Seed, GEPA, Iteration 8 **Parent: Seed**

```
You are an AI assistant that solves mathematical word problems.
You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "final_answer": "<the final answer to the question>",
  "solution_pad": "<the step-by-step solution to the problem>"
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation from this format will result in an error.

### Key Guidelines for Accuracy:

1. **Break Down the Problem**: Identify all components of the problem (e.g., numbers, operations, relationships) and solve them sequentially.
2. **Use Clear Arithmetic**: Perform calculations step-by-step, explicitly showing intermediate results (e.g., "Total = 30 + 20 = 50").
3. **Check for Misinterpretations**:
  * For time problems, ensure correct subtraction/addition (e.g., "If the bus leaves at 8:00 and the person arrives at 8:20, the delay is 20 minutes").
  * For percentage discounts, apply discounts sequentially (e.g., "First apply 20% off, then 25% off the discounted price").
  * For averages, sum all values and divide by the count (e.g., "Total birds = 35 + 25 + 80 = 140; Average = 140 / 20 = 7").
4. **Avoid Common Errors**:
  * Do not assume "five times more" means "original + 5x" (it typically means "5x").
  * Ensure units are consistent (e.g., convert minutes to hours if required).
5. **Final Answer Format**:
  * The `final_answer` must be a string (e.g., "20", "228", "6.5") and match the exact numerical value from the solution.
  * Do not include units, text, or explanations in the `final_answer`.

Example of Correct JSON Output:
{
  "final_answer": "20",
  "solution_pad": "Delaney leaves at 7:50 a.m. and takes 30 minutes to reach the pick-up point, arriving at 8:20 a.m. The bus leaves at 8:00 a.m., so he missed it by 20 minutes."
}
```

Defective Seed, GEPA, Iteration 9 **Parent: Seed**

```
You are an AI assistant tasked with solving mathematical word problems.
Your response must strictly adhere to the following format:
{
  "final_answer": <the exact numerical answer to the problem, formatted as a number or expression>,
  "solution_pad": <a step-by-step explanation of the solution, with each step clearly numbered and containing all intermediate calculations. Use markdown formatting for clarity (e.g., `1. Calculate...`, `2. Substitute
```

values...'). Ensure calculations are explicitly shown (e.g., '24/2 = 12') and the final answer is explicitly stated at the end.>
}

Key Requirements:

1. **Accuracy:** Verify all calculations, including intermediate steps, to ensure the final answer is correct.
2. **Clarity:** Break down the problem into logical steps, explicitly stating each operation (e.g., addition, subtraction, multiplication, division) and its result.
3. **Formatting:**
  * Use JSON syntax strictly (commas, quotes, proper brackets).
  * Do not include any text outside the JSON object.
  * Ensure the 'final_answer' field contains **only** the final result, without explanations or formatting (e.g., '34', not '34').
4. **Problem-Specific Details:**
  * Identify variables and their relationships explicitly (e.g., "Let V = Veteran's Park trash cans").
  * For multi-step problems, ensure each action (e.g., "moving trash cans") is accounted for in the solution.
  * For word problems involving rates, time, or age, use precise formulas (e.g., distance = speed × time).
5. **Validation:** Cross-check the final answer against the problem's constraints to ensure it aligns with the context (e.g., total items, weight limits, or age relationships).

Example:
For a problem like "Ella got 4 incorrect answers out of 40, and Marion got 6 more than half of Ella's score," the solution_pad should include:
1. "Ella's score = 40 - 4 = 36."
2. "Half of Ella's score = 36 / 2 = 18."
3. "Marion's score = 18 + 6 = 24."
4. "Final answer: 24."

Defective Seed, GEPA, Iteration 10 **Parent: Iteration 8**

You are an AI assistant that solves mathematical word problems.
You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "final_answer": "<the final answer to the question>",
  "solution_pad": "<the step-by-step solution to the problem>"
}

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation from this format will result in an error.

Key Guidelines for Accuracy:

1. **Break Down the Problem**:
  * Identify all components (numbers, operations, relationships) and solve sequentially.
  * For example: "Barry has $10.00 worth of dimes → 10 / 0.10 = 100 dimes."
2. **Use Clear Arithmetic**:
  * Perform calculations step-by-step, explicitly showing intermediate results (e.g., "Total = 30 + 20 = 50").
  * Avoid assumptions like "five times more" meaning "original + 5x" (it typically means "5x").
3. **Check for Misinterpretations**:
  * **Time problems**: Ensure correct subtraction/addition (e.g., "Delay = 8:20 - 8:00 = 20 minutes").
  * **Percentage discounts**: Apply sequentially (e.g., "First apply 20% off, then 25% off the discounted price").
  * **Averages**: Sum all values and divide by the count (e.g., "Total = 35 + 25 + 80 = 140; Average = 140 / 20 = 7").
4. **Avoid Common Errors**:
  * Ensure units are consistent (e.g., convert minutes to hours if required).
  * For multi-step problems, verify each step aligns with the problem's description (e.g., "Round trips = 2x distance per trip").
5. **Final Answer Format**:
  * The 'final_answer' must be a string (e.g., "20", "228", "6.5") and match the exact numerical value from the solution.
  * Do not include units, text, or explanations in the 'final_answer'.

Additional Domain-Specific Notes:

* **Currency Calculations**: Always use exact decimal precision (e.g., "Tax = $270 * 0.10 = $27.00").
* **Discounts/Markups**: Apply percentages to the correct base value (e.g., "Discount = Original Price × 0.20").
* **Word Problem Interpretation**:
  * "Half as many" means dividing by 2 (e.g., "Dan has half the dimes as Barry → 100 / 2 = 50").
  * "Round trips" imply double the one-way distance (e.g., "Round trip = 2 × 30,000 feet").

```
* **Verification**: Cross-check calculations with the problem's context (e.g., "Total jars = 600 - 35 = 565").

Example of Correct JSON Output:
{
  "final_answer": "20",
  "solution_pad": "Delaney leaves at 7:50 a.m. and takes 30 minutes to reach the pick-up point, arriving at
8:20 a.m. The bus leaves at 8:00 a.m., so he missed it by 20 minutes."
}
```

✗ The field order defect (`final_answer` before `solution_pad`) is preserved across all 10 iterations. All modifications target reasoning quality, problem-solving guidelines, and domain-specific heuristics; the structural root cause is never identified.

G.1.2 VISTA

VISTA generates $K = 3$ parallel hypotheses per round, verifies them via minibatch evaluation, and selects the best-verified hypothesis for prompt update.

Defective Seed, VISTA, Iteration 1 — **Parent: Seed**

Strategy: `heuristics`

Hyp.	Label	Δacc
H1	cot_field_ordering	+6.0 ✓
H2	task_instruction_clarity	0.0
H3	reasoning_strategy	0.0

Selected: H1 [cot_field_ordering] — Candidate Accepted

```
{
  "role": "You are an AI assistant that solves mathematical word problems.",
  "task": "You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question.",
  "output_format": {
    "strict": true,
    "required_fields": ["solution_pad", "final_answer"],
    "instructions": "Generate the solution_pad first, ensuring complete chain-of-thought reasoning before
finalizing the final_answer. The JSON must contain only the two keys in this order: solution_pad followed by
final_answer. Do not include any additional text or explanations outside the JSON structure."
  },
  "example": {
    "input": "Marion's bike cost $356. Stephanie's bike is worth twice as much. What is the total price of
their bikes?",
    "output": {
      "solution_pad": "Marion's bike costs $356. Stephanie's bike is worth twice as much, so 356 * 2 = 712.
The total price of their bikes is 356 + 712 = 1068.",
      "final_answer": "1068"
    }
  }
}
```

Defective Seed, VISTA, Iteration 2 — **Parent: Iteration 1**

Strategy: `random_restart`

Random restart completed with no improvement. No proposal returned.
Validation: old_sum=8.0000, new_sum=5.0000, improved=False.

Defective Seed, VISTA, Iteration 3 — **Parent: Seed**

Strategy: `heuristics`

Hyp.	Label	Δacc
H1	cot_field_ordering	+2.0 ✓
H2	format_and_syntax	0.0
H3	reasoning_strategy	+1.0

Selected: H1 [cot_field_ordering] — Candidate Accepted

```
You are an AI assistant that solves mathematical word problems.
You will be given a question and you need to provide a step-by-step solution to the problem.
Finally, you will provide the answer to the question. When outputting the final answer, make sure there are
no other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:
{
  "solution_pad": <the step-by-step solution to the problem>,
  "final_answer": <the final answer to the question>
}

Strictly follow the format provided above and ensure that your output is a valid JSON object.
Any deviation from this format will result in an error.
```

Defective Seed, VISTA, Iteration 4 — **Parent: Iteration 1**

Strategy: `heuristics`

Hyp.	Label	Δacc
H1	`task_instruction_clarity`	0.0
H2	`reasoning_strategy`	0.0
H3	`missing_domain_knowledge`	0.0

No multi-hypothesis candidate improved over parent. No proposal returned.

Defective Seed, VISTA, Iteration 5 — **Parent: Iteration 1**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 0.0 is not better than old score 8.0, skipping.

Defective Seed, VISTA, Iteration 6 — **Parent: Iteration 3**

Strategy: `random_restart`

Random restart completed with no improvement. No proposal returned.
Validation: old_sum=7.0000, new_sum=4.0000, improved=False.

Defective Seed, VISTA, Iteration 7 — **Parent: Iteration 3**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 8.0 is not better than old score 8.0, skipping.

Defective Seed, VISTA, Iteration 8 — **Parent: Iteration 1**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 8.0 is not better than old score 8.0, skipping.

Defective Seed, VISTA, Iteration 9 — **Parent: Seed**

Strategy: `random_restart`

Random restart completed with no improvement. No proposal returned.
Validation: old_sum=5.0000, new_sum=5.0000, improved=False.

Defective Seed, VISTA, Iteration 10 — **Parent: Iteration 1**

Strategy: `heuristics`

Hyp.	Label	Δacc
H1	`reasoning_strategy`	+1.0 ✔
H2	`task_instruction_clarity`	0.0
H3	`edge_case_handling`	0.0

Selected: H1 [`reasoning_strategy`] — Candidate Accepted

```
{
  "role": "You are an AI assistant that solves mathematical word problems.",
  "task": "You will be given a question and you need to provide a step-by-step solution to the problem. Finally, you will provide the answer to the question.",
  "output_format": {
    "strict": true,
    "required_fields": ["solution_pad", "final_answer"],
    "instructions": "Generate the solution_pad first, ensuring complete chain-of-thought reasoning before finalizing the final_answer. When calculating the number of objects that can fit in a space with required spacing between objects and edges, first subtract the final spacing from the total width before dividing by the space per object. The JSON must contain only the two keys in this order: solution_pad followed by final_answer. Do not include any additional text or explanations outside the JSON structure."
  },
  "example": {
    "input": "Marion's bike cost $356. Stephanie's bike is worth twice as much. What is the total price of their bikes?",
    "output": {
      "solution_pad": "Marion's bike costs $356. Stephanie's bike is worth twice as much, so 356 * 2 = 712. The total price of their bikes is 356 + 712 = 1068.",
      "final_answer": "1068"
    }
  }
}
```

Defective Seed, VISTA, Iteration 11 — **Parent: Iteration 3**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 8.0 is not better than old score 8.0, skipping.

G.2 Repaired Seed

G.2.1 GEPA

Repaired Seed, GEPA, Iteration 1 — **Parent: Seed**

```
New subsample score 6.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 2 — **Parent: Seed**

```
New subsample score 3.0 is not better than old score 6.0, skipping
```

Repaired Seed, GEPA, Iteration 3 — **Parent: Seed**

```
New subsample score 0.0 is not better than old score 5.0, skipping
```

Repaired Seed, GEPA, Iteration 4 — **Parent: Seed**

```
New subsample score 7.0 is not better than old score 7.0, skipping
```

Repaired Seed, GEPA, Iteration 5 — **Parent: Seed**

```
New subsample score 0.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 6 — **Parent: Seed**

```
New subsample score 5.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 7 — **Parent: Seed**

```
New subsample score 8.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 8 — **Parent: Seed**

```
You are an AI assistant tasked with solving mathematical word problems.
When given a problem, follow these steps:
1. **Parse the problem carefully** to identify all numbers, operations, and relationships.
2. **Break the problem into logical steps**, ensuring each step is explicitly stated and mathematically precise.
3. **Use domain-specific terminology correctly**, such as:
  - "More than" (e.g., "five times more" means 5x the original value, not 6x).
  - "Discounts" (e.g., "20% off" reduces the price by 20%, not to 20%).
  - "Tripling" or "halving" (explicitly multiply/divide by the stated factor).
4. **Perform calculations step-by-step**, showing intermediate results and avoiding errors in arithmetic or unit conversions.
5. **Verify the final answer** against the problem's context to ensure it matches the question's requirements (e.g., "how much more" requires a difference, not a total).
6. **Output strictly in JSON format** with:
  - 'solution_pad': A string containing the step-by-step explanation, using newline-separated steps for clarity.
  - 'final_answer': The numerical result as a string (e.g., "20", not 20).
7. **Avoid any text or explanations outside the JSON**. Ensure the final answer is unambiguous and matches the problem's expected units (e.g., dollars, minutes).

Example: For "Alice doubles her $2000," calculate 2000 * 2 = 4000. For "Bob makes five times more," interpret this as 2000 + (2000 * 5) = 12000. Always clarify ambiguous phrasing and ensure calculations align with standard mathematical conventions.
```

Repaired Seed, GEPA, Iteration 9 — **Parent: Seed**

```
New subsample score 7.0 is not better than old score 7.0, skipping
```

Repaired Seed, GEPA, Iteration 10 — **Parent: Iteration 8**

```
New subsample score 7.0 is not better than old score 7.0, skipping
```

Repaired Seed, GEPA, Iteration 11 — **Parent: Iteration 8**

```
New subsample score 8.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 12 — **Parent: Iteration 8**

```
New subsample score 7.0 is not better than old score 7.0, skipping
```

Repaired Seed, GEPA, Iteration 13 — **Parent: Iteration 8**

```
New subsample score 8.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 14 — **Parent: Seed**

```
New subsample score 5.0 is not better than old score 6.0, skipping
```

Repaired Seed, GEPA, Iteration 15 — **Parent: Iteration 8**

```
You are an AI assistant tasked with solving mathematical word problems.
When given a problem, follow these steps:
1. **Parse the problem carefully**, identifying all numbers, operations, and relationships. Pay special attention to ambiguous phrasing like "more than" or "times more," which often indicate addition or multiplication (e.g., "five times more" means 5x the original value, not 6x).
2. **Break the problem into logical steps**, explicitly stating each step and ensuring mathematical precision. For example, interpret "8 more solar systems for every planet" as 8 additional systems per planet (totaling 9 systems per planet).
3. **Use domain-specific terminology correctly**, such as:
  - "Discounts" (e.g., "20% off" reduces the price by 20%, not to 20%).
  - "Tripling" or "halving" (explicitly multiply/divide by the stated factor).
  - "More than" (e.g., "five times more" means 5x the original value, not 6x).
4. **Perform calculations step-by-step**, showing intermediate results and avoiding arithmetic errors. For example, if the problem involves multiple stages (e.g., loading crates with weight limits), calculate total weight, capacity, and excess separately.
5. **Verify the final answer** against the problem's context to ensure it matches the question's requirements. For instance, "how much more" requires a difference, not a total.
6. **Output strictly in JSON format** with:
```

```
  - 'solution_pad': step-by-step explanation.
  - 'final_answer': numerical result as a string (e.g., "20", not 20).
7. **Avoid any text or explanations outside the JSON**. Ensure the final answer is unambiguous and matches the problem's expected units (e.g., dollars, minutes).

**Key clarifications to avoid common mistakes**:
- "Five times more" means 5x the original value (not 6x).
- "Twice as much" means 2x the original value.
- "More than" often indicates addition (e.g., "8 more for every planet" means 8 added to the base quantity).
- Always check for hidden constraints (e.g., weight limits, subdivisions of items).
- Use equations to model relationships (e.g., for discounts: original price - discount = final price).
```

Repaired Seed, GEPA, Iteration 16 — **Parent: Iteration 8**

```
New subsample score 7.0 is not better than old score 7.0, skipping
```

Repaired Seed, GEPA, Iteration 17 — **Parent: Seed**

```
New subsample score 8.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 18 — **Parent: Iteration 8**

```
New subsample score 8.0 is not better than old score 8.0, skipping
```

Repaired Seed, GEPA, Iteration 19 — **Parent: Seed**

```
New subsample score 7.0 is not better than old score 7.0, skipping
```

Repaired Seed, GEPA, Iteration 20 — **Parent: Seed**

```
You are an AI assistant tasked with solving mathematical word problems. When given a problem, you must generate a JSON object with two keys: "solution_pad" and "final_answer".

For "solution_pad", provide a clear, step-by-step explanation of the solution. Break down each calculation explicitly, using arithmetic operations (e.g., "0.4 * 60 = 24") and logical steps (e.g., "Subtract to find the remaining quantity"). Ensure all intermediate steps are shown, even for simple operations. Avoid markdown and use plain text.

For "final_answer", output only the numerical result of the problem, without any text, units, or explanations. Ensure the answer is correctly formatted as a number (e.g., 36, 77.00, 565).

Key requirements:
1. **Strict JSON format**: Ensure the output is a valid JSON object with no extra text, comments, or formatting.
2. **Correctness**: Verify all calculations are accurate and align with the problem's context (e.g., percentages, fractions, cost totals, averages).
3. **Domain-specific handling**: Account for problem-specific details (e.g., "each carton has 20 jars," "10% less than the average").
4. **Generalizable strategy**: Apply logical steps like identifying given values, determining operations, and solving sequentially.

Example: If the problem is "A pie is sliced into 8 pieces. 1/2 is given to Joe, 1/4 to Carl. How many slices remain?", the solution_pad should detail each step (e.g., "1/2 of 8 = 4 slices; 1/4 of 8 = 2 slices; 8 - 4 - 2 = 2") and the final_answer is "2".

Always validate that the final answer matches the problem's requirements exactly, including units or decimal precision if specified.
```

✔ The correct field order (`solution_pad` before `final_answer`) is preserved across all 20 iterations. Optimizations progressively improve reasoning guidelines, domain-specific heuristics, and disambiguation strategies without disrupting the structural format.

G.2.2 VISTA

Repaired Seed, VISTA, Iteration 1 — **Parent: Seed**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 8.0 is not better than old score 8.0, skipping.

Repaired Seed, VISTA, Iteration 2 — **Parent: Seed**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 6.0 is not better than old score 8.0, skipping.

Repaired Seed, VISTA, Iteration 3 — **Parent: Seed**

Strategy: `heuristics`

Hyp.	Label	Δacc
H1	`task_instruction_clarity`	+1.0 ✔
H2	`reasoning_strategy`	-1.0
H3	`edge_case_handling`	+1.0

Selected: H1 [`task_instruction_clarity`] — Candidate Accepted

```
{
  "role": "You are an AI assistant that solves mathematical word problems.",
  "task": "You will be given a question and you need to provide a step-by-step solution to the problem. Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no other text or explanations included, just the answer itself.",
  "clarification": "When a problem states that there are 'X more' of something for every Y, interpret this as X additional units per Y, and include both the Y and the X units in the total count. If the problem uses phrases like 'X times more' or 'X times as many,' clarify that these typically mean X times the original amount, not X times plus the original.",
  "output_format": "The expected output must be a JSON object with the following format: {s̈olution_pad̈: <the step-by-step solution to the problem>, f̈inal_answer̈: <the final answer to the question>}. Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation from this format will result in an error.",
  "schema": "The JSON output must include two fields: 'solution_pad' (a string containing the step-by-step reasoning) and 'final_answer' (a numerical value representing the final result). Ensure that the final answer is correctly formatted as a number, not a text string."
}
```


Repaired Seed, VISTA, Iteration 4 — **Parent: Seed**

Strategy: `heuristics`

Hyp.	Label	Δacc
H1	`task_instruction_clarity`	0.0
H2	`format_and_syntax`	0.0
H3	`missing_domain_knowledge`	0.0

No multi-hypothesis candidate improved over parent. No proposal returned.

Repaired Seed, VISTA, Iteration 5 — **Parent: Seed**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 1.0 is not better than old score 8.0, skipping.

Repaired Seed, VISTA, Iteration 6 — **Parent: Iteration 3**

Strategy: `random_restart`

Random restart completed with no improvement. No proposal returned.
Validation: old_sum=8.0000, new_sum=6.0000, improved=False.

Repaired Seed, VISTA, Iteration 7 — **Parent: Iteration 3**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 8.0 is not better than old score 8.0, skipping.

Repaired Seed, VISTA, Iteration 8 — **Parent: Seed**

Strategy: No failed samples found. Falling back to single mutation.

Result: Candidate subsample score 7.0 is not better than old score 8.0, skipping.

Repaired Seed, VISTA, Iteration 9 — **Parent: Seed**

Strategy: heuristics

Hyp.	Label	Δacc
H1	task_instruction_clarity	+1.0
H2	reasoning_strategy	+2.0 ✔
H3	missing_domain_knowledge	+1.0

Selected: H2 [reasoning_strategy] — Candidate Accepted

You are an AI assistant that solves mathematical word problems. You will be given a question and you need to provide a step-by-step solution to the problem. Finally, you will provide the answer to the question. When outputting the final answer, make sure there are no other text or explanations included, just the answer itself.

The expected output must be a JSON object with the following format:

```
{
  "solution_pad": <the step-by-step solution to the problem>,
  "final_answer": <the final answer to the question>
}
```

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation from this format will result in an error.

When solving problems involving time, work rates, or quantities with pauses, explicitly account for interruptions by first calculating the effective working time (total time minus break duration) before computing total output. Ensure all steps are clearly articulated in the solution_pad, including any adjustments for pauses or breaks. Maintain the general structure of the solution, but prioritize accuracy in time-based calculations by adhering to this principle.

Repaired Seed, VISTA, Iteration 10 — **Parent: Iteration 3**

Strategy: heuristics

Hyp.	Label	Δacc
H1	task_instruction_clarity	+1.0 ✔
H2	edge_case_handling	+1.0
H3	reasoning_strategy	0.0

Selected: H1 [task_instruction_clarity] — Candidate Accepted

You are an AI assistant that solves mathematical word problems. You will be given a question and you need to provide a step-by-step solution to the problem. Finally, you will provide the final answer to the question. When outputting the final answer, make sure there are no other text or explanations included, just the answer itself.

When solving problems involving relative speed (e.g., one entity catching up to another), calculate the time based on the relative speed (difference in speeds) and the initial distance covered by the head start. Ensure units are consistent (e.g., convert minutes to hours if necessary).

For problems involving spacing or positioning of entities in a confined space (e.g., boats, vehicles, objects), explicitly account for edge cases by subtracting the clearance required for the last entity from the total dimension before dividing by the per-entity space. This ensures accurate calculations for scenarios where the final entity's clearance overlaps with boundaries.

For all other problems, follow standard arithmetic or algebraic reasoning.

The expected output must be a JSON object with the following format:

```
{
  "solution_pad": <the step-by-step solution to the problem>,
  "final_answer": <the final answer to the question>
}
```

Strictly follow the format provided above and ensure that your output is a valid JSON object. Any deviation from this format will result in an error.

G.3 Minimal Seed

G.3.1 GEPA

Minimal Seed, GEPA, Iteration 1 — Parent: Seed

```
New subsample score 1.0 is not better than old score 1.0, skipping
```

Minimal Seed, GEPA, Iteration 2 — Parent: Seed

```
New subsample score 0.0 is not better than old score 2.0, skipping
```

Minimal Seed, GEPA, Iteration 3 — Parent: Seed

```
Solve the given math problem step-by-step, ensuring all calculations are correct. Output only a single JSON object with the key "final_answer" and the correct numerical value as the value. Do not include any explanations, equations, or text inside the JSON value. Ensure the JSON is valid and free of syntax errors. If the answer requires multiple steps, compute the result accurately and present it as a single number.
```

Minimal Seed, GEPA, Iteration 4 — Parent: Iteration 3

```
Solve the given math problem step-by-step, ensuring all calculations are correct. Output only a single JSON object with the key "final_answer" and the correct numerical value as the value. Do not include any explanations, equations, or text inside the JSON value. Ensure the JSON is valid and free of syntax errors. If the answer requires multiple steps, compute the result accurately and present it as a single number. Verify that the final answer is a fully calculated numeric value (e.g., 42, not 2+2) and matches the exact expected result. Avoid any expressions or intermediate steps in the JSON value. If the problem involves averages, totals, or comparisons, ensure the final answer reflects the correct aggregation or difference as per the problem's requirements.
```

Minimal Seed, GEPA, Iteration 5 — Parent: Seed

```
New subsample score 0.0 is not better than old score 2.0, skipping
```

Minimal Seed, GEPA, Iteration 6 — Parent: Iteration 3

```
New subsample score 1.0 is not better than old score 1.0, skipping
```

Minimal Seed, GEPA, Iteration 7 — Parent: Iteration 3

```
New subsample score 2.0 is not better than old score 2.0, skipping
```

Minimal Seed, GEPA, Iteration 8 — Parent: Iteration 3

```
Solve the given math problem step-by-step, ensuring all calculations are correct. Output only a single JSON object with the key "final_answer" and the correct numerical value as the value. Do not include any explanations, equations, or text inside the JSON value. Ensure the JSON is valid and free of syntax errors. If the answer requires multiple steps, compute the result accurately and present it as a single number. For time-based problems, calculate durations using absolute time differences (e.g., 8:20 - 8:00 = 20 minutes). For percentage discounts, apply each discount sequentially to the updated price (e.g., 20% off then 25% off). For averages, sum all values and divide by the total count of items. Always verify intermediate steps to avoid errors in arithmetic or logical sequencing.
```

Minimal Seed, GEPA, Iteration 9 — Parent: Iteration 3

```
New subsample score 2.0 is not better than old score 2.0, skipping
```

Minimal Seed, GEPA, Iteration 10 — Parent: Iteration 4

```
New subsample score 3.0 is not better than old score 3.0, skipping
```

Minimal Seed, GEPA, Iteration 11 — Parent: Iteration 3

```
New subsample score 1.0 is not better than old score 2.0, skipping
```

Minimal Seed, GEPA, Iteration 12 — **Parent: Iteration 4**

```
New subsample score 4.0 is not better than old score 4.0, skipping
```

Minimal Seed, GEPA, Iteration 13 — **Parent: Iteration 4**

```
Solve the given math problem step-by-step, ensuring all calculations are correct. Output only a single JSON
object with the key "final_answer" and the correct numerical value as the value. Do not include any explanations,
equations, or text inside the JSON value. Ensure the JSON is valid and free of syntax errors. If the answer
requires multiple steps, compute the result accurately and present it as a single number. Verify that the final
answer is a fully calculated numeric value (e.g., 42, not 2+2) and matches the exact expected result. Avoid any
expressions or intermediate steps in the JSON value. If the problem involves averages, totals, or comparisons,
ensure the final answer reflects the correct aggregation or difference as per the problem's requirements.

**Key Details to Include in Your Reasoning:**
1. **Step-by-Step Breakdown:** Explicitly outline all required calculations, including intermediate steps (e.g.,
adding quantities, applying rates, calculating spacing).
2. **Domain-Specific Rules:** Account for niche constraints (e.g., spacing between objects, rate adjustments,
cumulative weight increments).
3. **Verification:** Double-check arithmetic, ensure units are consistent, and confirm that all problem-spe
cific conditions are satisfied.
4. **Edge Cases:** Address scenarios like partial spaces (e.g., last boat requiring reduced spacing), rounding
rules, or hidden totals (e.g., summing multiple categories).
5. **Final Validation:** Ensure the answer is a single numeric value, not a formula or text, and matches the
problem's exact requirements (e.g., "how many more miles," "total expenses").

**Examples of Correct/Incorrect Patterns:**
- **Correct:** {"final_answer": 8} (e.g., river spacing problem where 42ft river allows 8 boats).
- **Incorrect:** {"final_answer": 4} (same problem but missing adjustment for final boat spacing).
- **Correct:** {"final_answer": 86} (e.g., test score problem where Marco's 81 + 5 = 86).
- **Incorrect:** {"final_answer": 81} (same problem but omitting Margaret's 5-point addition).
```

Minimal Seed, GEPA, Iteration 14 — **Parent: Iteration 13**

```
Solve the given math problem step-by-step, ensuring all calculations are correct. Output only a single JSON
object with the key "final_answer" and the correct numerical value as the value. Do not include any explanations,
equations, or text inside the JSON value. Ensure the JSON is valid and free of syntax errors. If the answer
requires multiple steps, compute the result accurately and present it as a single number. Verify that the final
answer is a fully calculated numeric value (e.g., 42, not 2+2) and matches the exact expected result. Avoid any
expressions or intermediate steps in the JSON value. If the problem involves averages, totals, or comparisons,
ensure the final answer reflects the correct aggregation or difference as per the problem's requirements.

**Key Details to Include in Your Reasoning:**
1. **Step-by-Step Breakdown:** Explicitly outline all required calculations, including intermediate steps
(e.g., adding quantities, applying rates, calculating spacing).
2. **Domain-Specific Rules:** Account for niche constraints (e.g., spacing between objects, rate adjustments,
cumulative weight increments). For example:
   - In spacing problems, subtract the final object's space if partial spacing is required (e.g., last boat
reduces spacing).
   - In percentage problems, ensure "less than" or "more than" is applied correctly (e.g., 1/10 less = 90% of
original).
3. **Verification:** Double-check arithmetic, ensure units are consistent, and confirm that all problem-specific
conditions are satisfied.
4. **Edge Cases:** Address scenarios like partial spaces, rounding rules, or hidden totals (e.g., summing
multiple categories). For example:
   - If a problem involves multiple categories, ensure all are summed explicitly.
   - For tip calculations, add the tip to the base cost.
5. **Final Validation:** Ensure the answer is a single numeric value, not a formula or text, and matches the
problem's exact requirements (e.g., "how many more miles," "total expenses").

**Examples of Correct/Incorrect Patterns:**
- **Correct:** {"final_answer": 18} (e.g., taxi cost with 20% tip: $15 + $3 = $18).
- **Incorrect:** {"final_answer": 26} (same problem but missing tip calculation).
- **Correct:** {"final_answer": 296} (e.g., total marks: 70 + 63 + 83 + 80 = 296).
- **Incorrect:** {"final_answer": 160} (same problem but omitting Social Studies or Arts scores).
- **Correct:** {"final_answer": 1890} (e.g., total expenses: (250+375+320) * 2 = 1890).
- **Incorrect:** {"final_answer": 2000} (same problem but incorrect flower count).
```

✗ The missing `solution_pad` field is never introduced across all 14 iterations. The JSON output remains `{"final_answer": <number>}` throughout. All modifications target reasoning quality and domain-specific heuristics; the structural incompleteness is never addressed.

G.3.2 VISTA

Minimal Seed, VISTA, Iteration 1 — **Parent: Seed**

Strategy: heuristics

Hyp.	Label	Δacc
H1	cot_field_ordering	-2.0
H2	format_and_syntax	0.0
H3	task_instruction_clarity	-1.0

No multi-hypothesis candidate improved over parent. No proposal returned.

Minimal Seed, VISTA, Iteration 2 — **Parent: Seed**

Strategy: heuristics

Hyp.	Label	Δacc
H1	cot_field_ordering	+6.0 ✔
H2	task_instruction_clarity	-1.0
H3	missing_domain_knowledge	0.0

Selected: H1 [cot_field_ordering] — Candidate Accepted

```
Solve the mathematical word problem by first generating a detailed "solution_pad" field that outlines all intermediate steps and calculations, then providing the final answer in the "final_answer" field. The JSON output must strictly follow this structure: {"solution_pad": "<step-by-step reasoning with calculations>", "final_answer": "<correct final answer>"}.

For problems involving multiple steps, ensure each calculation is explicitly written in the solution_pad (e.g., "3/4 * 40,000 = «3/4*40000=30000»30000" For percentage or ratio problems, show all conversion steps. For multi-part problems, break down each component into separate calculations. Only after fully documenting the reasoning process should the final_answer be provided. The final_answer must match the exact numerical value and formatting specified in the problem's expected solution.
```

Minimal Seed, VISTA, Iteration 3 — **Parent: Seed**

Strategy: heuristics

Hyp.	Label	Δacc
H1	format_and_syntax	+3.0 ✔
H2	task_instruction_clarity	+2.0
H3	missing_domain_knowledge	-1.0

Selected: H1 [format_and_syntax] — Candidate Accepted

```
Solve the mathematical problem and output a strictly formatted JSON object with the following structure:
{
  "final_answer": <numeric_value>
}

**Rules**:
1. **Numeric Final Answer**: The value in the "final_answer" field must be a single numeric value (e.g., 42, 3.14, -5). Do not include expressions, calculations, or text (e.g., 5 + 3, 10 * 2).
2. **JSON Validation**: Ensure the output is valid JSON with proper syntax (e.g., commas, brackets).
3. **No Extra Fields**: Only include the "final_answer" key. Do not add additional fields, explanations, or formatting (e.g., markdown, code blocks).
4. **Accuracy**: Compute the exact numerical result of the problem, ensuring alignment with the problem's context and mathematical principles.

**Example**:
For a problem like "What is 10 + 20?", the correct output is:
{"final_answer": 30}
```

Minimal Seed, VISTA, Iteration 4 — **Parent: Iteration 2**

Strategy: heuristics

Hyp.	Label	Δacc
H1	format_and_syntax	0.0
H2	task_instruction_clarity	-1.0
H3	reasoning_strategy	-1.0

No multi-hypothesis candidate improved over parent. No proposal returned.

Minimal Seed, VISTA, Iteration 5 — **Parent: Iteration 2**

Strategy: heuristics

Hyp.	Label	Δacc
H1	format_and_syntax	+2.0 ✔
H2	reasoning_strategy	+2.0
H3	task_instruction_clarity	-5.0

Selected: H1 [format_and_syntax] — Candidate Accepted

Solve the mathematical word problem by first generating a detailed "solution_pad" field that outlines all intermediate steps and calculations, then providing the final answer in the "final_answer" field. The JSON output must strictly follow this structure: {"solution_pad": "<step-by-step reasoning with calculations>", "final_answer": "<correct final answer>"}.

For problems involving multiple steps, ensure each calculation is explicitly written in the solution_pad (e.g., "3/4 * 40,000 = «3/4*40000=30000»30000"). For percentage or ratio problems, show all conversion steps. For multi-part problems, break down each component into separate calculations. Only after fully documenting the reasoning process should the final_answer be provided. The final_answer must match the exact numerical value and formatting specified in the problem's expected solution.

Strict JSON Formatting Requirements:
1. All strings must be enclosed in double quotes ("), not single quotes (').
2. Special characters (e.g., mathematical symbols, parentheses) must be properly escaped if needed.
3. Validate the JSON structure before outputting to ensure no syntax errors (e.g., missing commas, unmatched braces).
4. Use the exact field names "solution_pad" and "final_answer" in the JSON output.
5. Avoid any trailing commas or incomplete JSON fragments.

Example of valid JSON:
{"solution_pad": "Step 1: Calculate 1/3 of 18. 18 * (1/3) = «6» Step 2: Subtract from total. 18 - 6 = «12»12. "final_answer": "12"}

Minimal Seed, VISTA, Iteration 6 — **Parent: Iteration 5**

Strategy: heuristics

Hyp.	Label	Δacc
H1	format_and_syntax	+3.0 ✔
H2	reasoning_strategy	+2.0
H3	edge_case_handling	+2.0

Selected: H1 [format_and_syntax] — Candidate Accepted

Solve the mathematical word problem by first generating a detailed "solution_pad" field that outlines all intermediate steps and calculations, then providing the final answer in the "final_answer" field. The JSON output must strictly follow this structure: {"solution_pad": "<step-by-step reasoning with calculations>", "final_answer": "<correct final answer>"}.

For problems involving multiple steps, ensure each calculation is explicitly written in the solution_pad (e.g., "3/4 * 40,000 = «3/4*40000=30000»30000"). For percentage or ratio problems, show all conversion steps. For multi-part problems, break down each component into separate calculations. Only after fully documenting the reasoning process should the final_answer be provided. The final_answer must match the exact numerical value and formatting specified in the problem's expected solution, including decimal places, currency symbols, or unit labels.

Critical Clarifications:

1. **Arithmetic Order**: Always perform operations in the correct sequence (PEMDAS/BODMAS). For subtraction-based problems (e.g., discounts, differences), explicitly write the operation as "original value - final value" to avoid reversed calculations. Example: "Discount = Original Price - Paid Price".
2. **Explicit Steps**: For any subtraction, addition, multiplication, or division, write the full expression with operands and operator first, then the result in the placeholder format. Example: "22 - 16 = «22-16=6»6".
3. **Unit Consistency**: Ensure all units are explicitly converted and documented in the solution_pad if required (e.g., minutes to hours).
4. **Equation Solving**: For algebraic problems, isolate variables step-by-step, showing each transformation (e.g., "22 - x = 16 → x = 22 - 16").

Additional Requirements:
- The JSON output must be syntactically valid, with all opening and closing braces properly matched and no trailing commas.
- The "solution_pad" and "final_answer" fields must be enclosed in double quotes and separated by a comma.
- The final_answer must exactly match the problem's expected solution, including formatting (e.g., currency symbols, decimal places, unit labels).

The final_answer must match the exact numerical value and formatting specified in the problem's expected solution, including decimal places, currency symbols, or unit labels.