DePRO: Stable Prompt Optimization by Editing One Module at a Time

Anonymous ACL submission

Abstract

Existing prompt optimization methods rewrite prompts as monolithic text, making performance changes non-attributable: when accuracy shifts, it is unclear which functional change—formatting, reasoning constraints, or grounding instructions—caused the effect. We propose DePRO (Decoupled Prompt Refinement), a framework that decomposes prompts into a fixed base instruction plus named functional modules. We formalize three criteria for valid modules—semantic independence, replaceability, and effect attributability—and verify them through module ablations. During optimization, DePRO applies controlled-variable intervention: each iteration rewrites exactly one module while keeping others fixed, establishing a traceable causal chain from modification to performance change. A monotonic acceptance mechanism, GateON, accepts updates only when validation scores improve, enabling rollback and diagnostic logging. On GSM8K hard-top300, DePRO+GateON reduces optimization variance from 0.0051 to 0.0019 and reveals cooperative effects between exact-format and concise-answer modules, plus negative interference from evidence-support—interaction patterns hidden under whole-prompt rewriting. DePRO reframes prompt optimization as a formal, attributable process for studying functional module dependencies.

1 Introduction

Prompt optimization has become essential for adapting large language models to specific tasks. However, existing methods face a fundamental scientific problem: **when a prompt contains multiple functional constraints—such as output formatting rules, reasoning strategies, and safety guidelines—how can we formally define these functions so that their individual effects on performance can be isolated and measured?**

This is not merely an engineering challenge of avoiding bad updates. It is a scientific question about establishing causal attribution in prompt optimization. Current methods treat prompts as monolithic text blocks and rewrite them wholesale at each iteration (Yang et al., 2024a; Pryzant et al., 2023; Zhou et al., 2023; Prasad et al., 2023; Fernando et al., 2024; Yang et al., 2024b; Wang et al., 2024; Agrawal et al., 2026). For example, OPRO (Yang et al., 2024a) uses the LLM itself to generate improved prompts based on previous attempts and their scores. ProTeGi (Pryzant et al., 2023) creates gradient-like textual feedback and applies it to the entire prompt. APE (Zhou et al., 2023) samples multiple candidate instructions and selects the best by likelihood. GrIPS (Prasad et al., 2023) performs edit-based search over full instructions, while Promptbreeder (Fernando et al., 2024) and GEPA (Agrawal et al., 2026) use evolutionary and reflective strategies.

While these methods can achieve high peak accuracy, they suffer from functional coupling: a single rewrite simultaneously changes formatting rules, reasoning instructions, and output constraints. When performance improves, we cannot determine which change was responsible. When performance drops, the failed candidate is discarded without understanding why it failed.

This functional coupling problem is compounded by prompt sensitivity: small phrasing changes can cause large performance swings (Ma et al., 2024; Sclar et al., 2024; Lu et al., 2022; Xiang et al., 2024). When a whole-prompt rewrite improves performance, we cannot tell whether the gain came from better reasoning instructions, cleaner output formatting, or simply rephrasing. When performance drops, we lose the opportunity to learn which specific functional change caused the failure.

The core scientific challenge is this: **how do we formally define prompt functional modules such that (1) each module encodes a single behavioral constraint, (2) modules can be modified indepen-**

dently, and (3) the performance effect of modifying one module can be measured in isolation from others? Solving this problem would transform prompt optimization from an opaque trial-and-error process into a systematic method where each change is attributable to a specific function.

We address this problem with **DEPRO** (Decoupled Prompt Refinement), a framework that decomposes prompts into named functional modules and optimizes one module at a time. Each module encodes a single behavioral constraint—such as output formatting, answer conciseness, or evidence grounding. By restricting each optimization step to modifying exactly one module while holding all others fixed, DEPRO establishes a direct causal link between the modified module and any resulting performance change.

To prevent performance degradation, DEPRO employs **GATEON**, a non-negative acceptance rule that only accepts updates when they do not harm performance. Rejected updates are preserved as diagnostic records, revealing which module modifications are unsafe under the current prompt configuration.

We evaluate DEPRO on GSM8K hard-top300, a challenging subset of mathematical reasoning problems. Our experiments yield three main findings:

1. **Stability through controlled updates**: GATEON reduces full-budget standard deviation by nearly 10× compared to global rewriting (std 0.0019 vs 0.0184 on full evaluation), demonstrating that modular refinement with rollback protection provides substantially more reliable optimization trajectories.

2. **Module interactions are measurable**: By editing modules in isolation, we observe that the *exact-format* and *concise-answer* modules cooperate effectively, while *evidence-support* consistently interferes with their gains—a finding impossible to detect with whole-prompt rewriting.

3. **Tradeoff between stability and peak performance**: Global rewriting achieves higher absolute accuracy (0.3787 vs 0.2413), confirming that DEPRO trades some peak performance for interpretability and stability. This makes DEPRO complementary to, rather than a replacement for, unconstrained optimization methods.

Contributions. (1) We formalize the functional module decomposition problem and propose three orthogonality criteria (semantic independence, replaceability, effect attributability) for defining valid modules; (2) DEPRO, a modular refinement framework that restricts each update to a single named module, enabling causal attribution of performance changes; (3) GATEON, a rollback-gated acceptance rule that prevents performance degradation and preserves rejected updates as diagnostic evidence; (4) Controlled experiments demonstrating that module identity and interaction structure measurably affect optimization stability, with nearly 10× lower full-budget standard deviation than global rewriting.

2 Related Work

Prompt sensitivity and stability. LLM behavior is sensitive to prompt surface forms—demonstration order (Xiang et al., 2024; Lu et al., 2022), instruction phrasing (Weller et al., 2024), and minor formatting choices can cause substantial accuracy swings (Min et al., 2022; Sclar et al., 2024). This fragility motivates optimization methods that make attributable changes rather than large uncontrolled rewrites.

Automatic prompt optimization. Recent methods optimize prompts with textual gradients, edit search, evolutionary search, and multi-branch exploration (Yang et al., 2024a; Pryzant et al., 2023; Prasad et al., 2023; Fernando et al., 2024; Wang et al., 2024; Yang et al., 2024b; Agrawal et al., 2026). All operate on prompts as monolithic units, making post-hoc attribution difficult. DEPRO differs by restricting each step to one named module, enabling traceable causal attribution.

Iterative refinement and rollback. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use feedback to iteratively revise outputs, but intrinsic self-correction is unreliable without external validation (Huang et al., 2024). Sequential model editing studies parameter-level interference (Zhang et al., 2024; Mitchell et al., 2022), but targets model weights rather than prompts and lacks rollback protection.

3 Method

DEPRO addresses the functional coupling problem through three design choices: (1) decompose

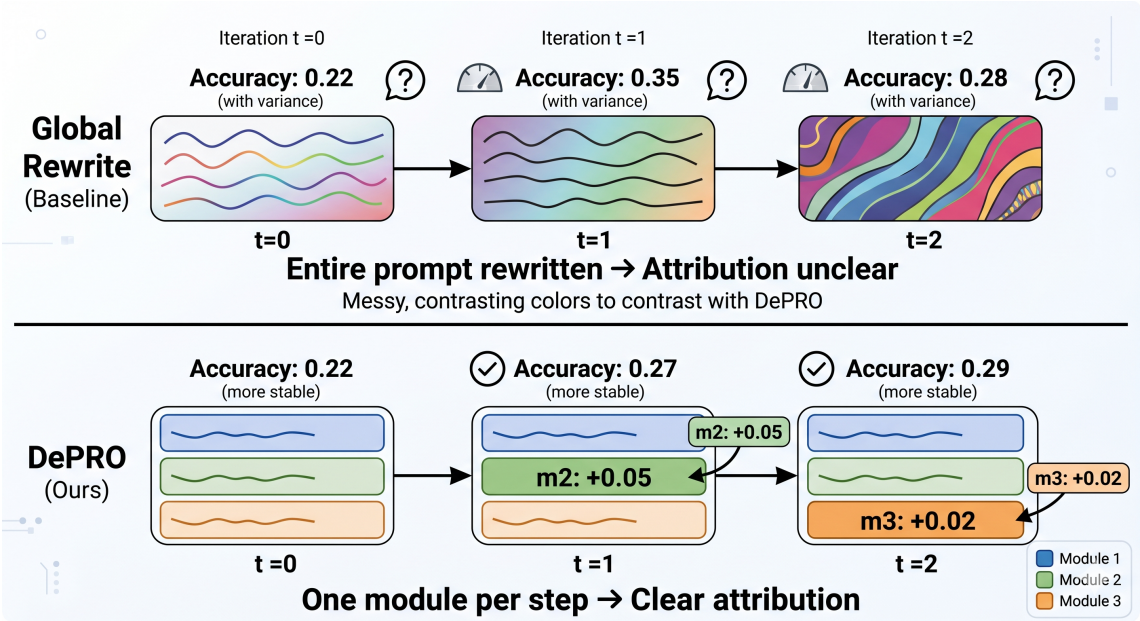

Figure 1: **Functional coupling problem in prompt optimization.** Existing methods modify multiple functional constraints simultaneously, making it impossible to attribute performance changes to specific components.

prompts into named functional modules with formal orthogonality criteria, (2) restrict each optimization step to modifying one module while holding others fixed, and (3) apply GATEON to prevent performance degradation.

3.1 Problem Formulation

The functional coupling problem. Existing prompt optimization methods treat prompts as monolithic text and modify multiple behavioral constraints simultaneously. When a rewrite changes output formatting, reasoning instructions, and safety guidelines all at once, we cannot determine which change caused the observed performance shift. This lack of causal attribution makes it impossible to systematically improve prompts or understand why certain modifications fail.

Formal definition of functional modules. We formalize a prompt as a fixed base instruction combined with a set of named functional modules:

$$P = P_{\text{base}} \oplus \{m_1, m_2, \ldots, m_K\}. \quad (1)$$

Each module m_k must satisfy three orthogonality criteria:

1. **Semantic independence**: Module m_k encodes a single behavioral constraint (e.g., output formatting, reasoning strategy) that does not overlap with other modules' functions.
2. **Replaceability**: Module m_k can be independently modified or removed without breaking the prompt's syntactic structure or requiring changes to other modules.
3. **Effect attributability**: The performance change caused by modifying m_k can be measured in isolation through ablation experiments, with minimal interaction effects from other modules.

These criteria ensure that when we modify module m_k and observe a performance change ΔS, we can confidently attribute ΔS to the modification of m_k rather than to confounding changes in other modules.

3.2 Prompt Modularization

Given the formal definition above, we now describe how to decompose a prompt into modules that satisfy the orthogonality criteria.

Module identification. We identify functional modules through a combination of semantic analysis and domain knowledge. For mathematical reasoning tasks, we observe three common failure modes: (1) hallucinated reasoning not grounded in the question, (2) malformed output that fails exact-match evaluation, and (3) verbose explanations that obscure the final answer. Each failure mode corresponds to a distinct behavioral constraint that can be addressed independently.

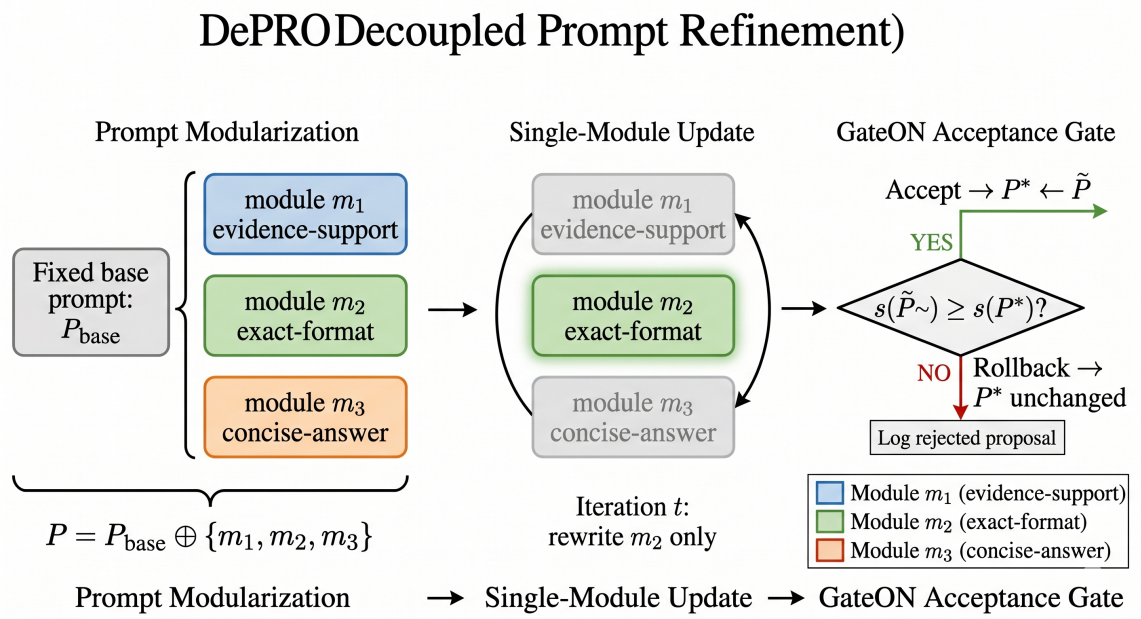

Figure 2: **DePRO framework overview.** Prompts are decomposed into named functional modules, with each optimization step modifying only one module while holding others fixed. GateON ensures non-degrading updates.

Module definitions for GSM8K. In our implementation, the base instruction is fixed as: *"You are a precise math solver. Output only the final numeric answer."* We define three modules targeting the observed failure modes: (1) **evidence-support** (m_1): "Answer must be directly supported by question content." (targets hallucination) (2) **exact-format** (m_2): "Prefer exact numeric or boolean output." (targets malformed output) (3) **concise-answer** (m_3): "Avoid verbose rationale in final answer." (targets verbosity)

These modules are designed to be semantically independent: each addresses a different aspect of model behavior. They are replaceable: any module can be modified or removed without affecting the others. Their effect attributability will be verified experimentally in Section 5. Full module strings are provided in Appendix B.

Verification of orthogonality. To verify that our modules satisfy the orthogonality criteria, we conduct ablation experiments (Section 5) measuring the interaction between module pairs. If modules m_i and m_j are truly orthogonal, their combined effect should be approximately additive: modifying both together should produce a performance change close to the sum of their individual effects.

3.3 Decoupled Refinement Loop

At each iteration t, DEPRO maintains a retained best prompt P^*_{t-1} and selects one active module m_k from the active set $\mathcal{A} \subseteq \{m_1, \ldots, m_K\}$. The loop is intentionally simple: evaluate the retained prompt, collect failure cases, rewrite the selected module, evaluate the candidate prompt, and keep the candidate only if it passes GATEON. Appendix D gives the reproducible step-by-step procedure.

By restricting each update to a single module, DEPRO ensures that any change in evaluation score is attributable to the module that was modified. Modules not in $\mathcal{A}$ are held constant throughout optimization.

Module selection strategy. We use round-robin scheduling to ensure each active module receives equal optimization budget. Alternative strategies include priority-based selection (focusing on modules with recent successful updates) or adaptive selection (allocating more budget to high-variance modules). We leave exploration of these strategies to future work.

Rewrite operator. Our rewrite operator is deliberately simple: given error cases and the current module text, we append a refinement clause (e.g., "be stricter about explicit evidence"). This simplicity isolates the effect of modular scheduling and gating from the sophistication of the proposal generator. More powerful rewrite operators (e.g., using stronger LLMs or retrieval-augmented generation) could improve absolute performance while preserv-

Configuration	Active set $\mathcal{A}$	GATEON
DePRO (baseline)	$\{m_1, m_2, m_3\}$	×
DePRO + GATEON (all)	$\{m_1, m_2, m_3\}$	✓
DePRO + GATEON (exact+concise)	$\{m_2, m_3\}$	✓

Table 1: Configurations compared in the ablation study. m_1=evidence-support, m_2=exact-format, m_3=concise-answer.

ing DEPRO's attribution and stability guarantees.

3.4 Non-Negative Acceptance Gate (GATEON)

To prevent destructive updates from overwriting hard-won prompt states, we introduce GATEON, a non-negative acceptance rule. Let $s(\tilde{P}_t)$ denote the score of the candidate and $s(P^*_{t-1})$ the score of the current best. GATEON accepts the update if and only if:

$$s(\tilde{P}_t) \;\geq\; s(P^*_{t-1}). \tag{2}$$

If the condition holds, $P^*_t \leftarrow \tilde{P}_t$ (accept); otherwise $P^*_t \leftarrow P^*_{t-1}$ (rollback), and the rejected proposal is preserved as a diagnostic record.

This design has two consequences. First, the retained prompt state is *monotonically non-degrading*: the best score never decreases due to an accepted update. Second, rejected proposals are not silently discarded; they constitute an interpretable record of *which module update directions are unsafe* under the current prompt context.

3.5 Active-Module Ablation Protocol

To study the contribution of individual modules, we vary only the active set $\mathcal{A}$ while holding all other factors constant: model endpoint, decoding configuration, sample budget, seeds, and evaluation order.

This protocol makes it possible to observe module *cooperation* (when a subset outperforms the full set) and module *antagonism* (when adding a module degrades performance)—phenomena invisible to holistic rewriting methods.

3.6 Complexity

DEPRO adds one evaluation call per module update proposal. With T iterations, $|\mathcal{A}|$ active modules, and batch size B, the total evaluation calls are $O(T \cdot |\mathcal{A}| \cdot B)$—the same asymptotic order as global rewriting. GATEON adds no model calls; it is a score comparison applied after each evaluation.

4 Experimental Setup

4.1 Task and Data

We evaluate on **GSM8K hard-top300**, a fixed held-out subset of 300 problems selected from the hardest examples in the GSM8K test split (Cobbe et al., 2021). GSM8K is a standard mathematical reasoning benchmark consisting of grade-school math word problems requiring multi-step arithmetic reasoning. Recent symbolic variants expose substantial fragility under controlled perturbations (Mirzadeh et al., 2025), suggesting that performance on this benchmark is sensitive to both reasoning quality and output formatting reliability.

Why GSM8K hard-top300? We focus on the hardest 300 examples for three reasons. First, these problems amplify sensitivity to prompt formatting and output serialization, making them well-suited for studying optimization stability. Second, the fixed subset ensures all methods are evaluated on identical instances, eliminating sampling variance. Third, the smaller size enables dense seed coverage (8 seeds for primary experiments, 3 seeds for full-budget validation) within reasonable compute budgets.

All methods are evaluated on the same instances; for each seed, sample ordering is shared across methods to ensure fair comparison. We use exact-match accuracy as the primary metric: a prediction is correct if and only if the final numeric answer matches the gold answer exactly.

4.2 Model and Runtime

All methods use the same **4B model endpoint** (Qwen3-4B-Thinking-2507) with identical decoding configuration: temperature 0.0, max output tokens 512, 60-second timeout. We chose a 4B model rather than a larger one for two reasons. First, smaller models exhibit higher prompt sensitivity, making optimization effects more observable. Second, the 4B size enables rapid iteration during method development and ablation studies.

Runtime settings are kept constant to isolate the effect of optimization strategy rather than compute budget. Each optimization run performs 3 iterations with round-robin module selection. We deliberately keep the iteration count low to study early-stage optimization dynamics, where the risk of destructive updates is highest.

Method	Description
Manual	Fixed human-written prompt, no optimization
Global Rewrite	Full-prompt rewrite at each iteration
DEPRO (baseline)	Modular refinement, no acceptance gating
DEPRO+GATEON(all)	All three modules active with rollback
DEPRO+GATEON(e+c)	Only exact-format and concise-answer active

Table 2: Methods compared. e+c = exact+concise.

Method	Mean EM ↑	Std ↓
Manual	0.2462	0.0362
Global Rewrite	**0.3787**	0.0511
DEPRO (baseline)	0.2263	0.0346
DEPRO+GATEON(all)	0.2413	**0.0336**
DEPRO+GATEON(e+c)	0.2350	0.0481

Table 3: **Main results** on GSM8K hard-top300, 100 samples × 8 seeds.

Method	Mean EM ↑	Std ↓
Manual	0.2189	0.0038
Global Rewrite	**0.3656**	0.0184
DEPRO (baseline)	0.2044	0.0051
DEPRO+GATEON(all)	0.2111	**0.0019**
DEPRO+GATEON(e+c)	0.2078	0.0038

Table 4: **Robustness results**, full top300 × 3 seeds.

4.3 Methods Compared

Baseline: Manual prompt. The manual baseline uses a fixed human-written prompt: *"You are a precise math solver. Output only the final numeric answer."* This baseline establishes the performance floor without any optimization.

Global Rewrite. At each iteration, Global Rewrite generates a completely new prompt by prompting an LLM with the current prompt, error cases, and the instruction to produce an improved version. This represents the standard approach in prior work (Yang et al., 2024a; Pryzant et al., 2023): the entire prompt is treated as a single optimization target.

DEPRO variants. DEPRO (baseline) applies modular refinement without acceptance gating: every proposed update is accepted regardless of performance change. DEPRO+GATEON adds the non-negative acceptance gate, rolling back updates that degrade performance. We compare two active-set configurations: (all) uses all three modules, while (e+c) uses only exact-format and concise-answer, excluding evidence-support.

4.4 Evaluation Protocol

- **Primary view**: 100 samples × 8 seeds — main accuracy and variance estimates.
- **Robustness view**: full top300 × 3 seeds — verifies results are not small-budget artifacts.
- **Pilot evidence** (appendix only): original 100 × 3 module ablation results.

Why multiple evaluation views? The primary view (100×8) balances sample size and seed diversity, providing stable mean estimates while capturing cross-seed variance. The robustness view (300×3) uses the full evaluation set to verify that findings generalize beyond the 100-sample subset, trading seed count for sample coverage. The pilot view (100×3) represents our initial exploration; we report it in the appendix for transparency but do not draw strong conclusions from it due to limited seed coverage.

We report mean exact-match accuracy and cross-seed standard deviation. Statistical comparisons use seed-level paired deltas and bootstrap 95% CIs, reported as **trend-level evidence** given the small seed count. We do not claim statistical significance; instead, we focus on effect magnitudes and consistency across evaluation views.

5 Results

5.1 Main Results (RQ1)

Table 3 shows that **Global Rewrite achieves the highest absolute accuracy** (0.3787). This result should not be obscured: DEPRO does not replace Global Rewrite as the strongest optimizer in terms of peak performance. Within the DEPRO family, **GATEON(all-modules) is the strongest supported variant**: it improves over baseline DEPRO in both mean accuracy (+0.0150) and cross-seed stability (std 0.0346 → 0.0336).

5.2 Robustness Under Full Budget (RQ1)

Table 4 confirms the DEPRO family ranking under the heavier evaluation. The variance reduction is most pronounced here: GATEON(all-modules) achieves std 0.0019 versus Global Rewrite's 0.0184—a **10× stability advantage**. This confirms that rollback-gated modular refinement provides a substantially more reliable optimization path.

5.3 Module Interaction Analysis (RQ2)

Table 5 reveals two findings. (1) **Cooperation**: the *exact-format* and *concise-answer* modules out-

Active set	Mean EM	Std	Δ vs all
all-modules	0.2700	0.0100	—
exact-only	0.2100	0.0265	−0.0600
exact+concise	**0.2800**	0.0265	+0.0100
exact+concise+evidence	0.2300	0.0400	−0.0400

Table 5: **Module ablation**, pilot 100 × 3 setting (exploratory evidence only).

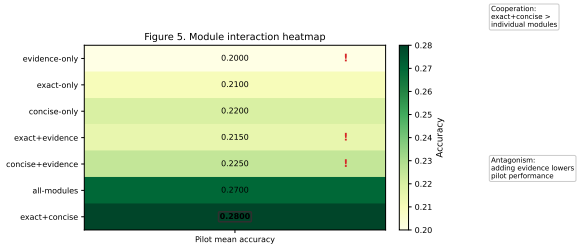

Figure 3: **Module interaction heatmap.** The heatmap shows performance changes when combining different module subsets. Darker colors indicate stronger positive interactions. The exact-format and concise-answer modules show cooperative effects, while evidence-support exhibits antagonistic interactions.

perform any single-module or full-set configuration. (2) **Antagonism**: adding *evidence-support* to exact+concise drops accuracy by 0.0500 and increases variance, suggesting conflicting optimization pressure.
Note: the exact+concise advantage did not stably replicate in the 100×8 expanded view (Table 3), so it is treated as exploratory evidence only.

5.4 GateON Rollback Mechanism (RQ3)

GATEON accepts only 33% of proposed updates on average, rolling back 67%. Crucially, **mean best score consistently exceeds mean last-iteration score** across all variants (Table 6), confirming that GATEON's value comes from *preserving stronger intermediate states*, not from finding better updates on the final step.

Qualitative analysis of rejected proposals. We manually inspected rejected module updates from one pilot run. Common rejection patterns include: (1) **Over-specification**: adding overly strict constraints that cause the model to refuse answering (e.g., "Answer must cite exact line numbers from the question"). (2) **Conflicting directives**: introducing rules that contradict existing modules (e.g., adding "Provide detailed step-by-step reasoning" when concise-answer already demands brevity). (3) **Formatting brittleness**: requiring output formats the model cannot reliably produce (e.g., "Output must be valid JSON with keys 'answer' and 'confidence'"). These rejection patterns suggest that GATEON acts as a *safety filter* against common failure modes in automated prompt rewriting.

Variant	Accept	Rollback	Best	Last-iter
GATEON(all), pilot	33.3%	66.7%	0.2700	0.2367
GATEON(e+c), pilot	55.6%	44.4%	0.2800	0.2233
GATEON(all), full300	33.3%	66.7%	0.2111	0.1889

Table 6: **Rollback summary.** Mean best score consistently exceeds mean last-iteration score, confirming GATEON preserves stronger intermediate states.

5.5 Case Study: Optimization Trajectory Analysis

We trace one representative run from each method to illustrate their differences.

Global Rewrite (seed 2026). Starting from baseline (acc 0.24), Global Rewrite improves to 0.38 (+0.14) then 0.41 (+0.03), but iteration-3 degrades to 0.35 (-0.06) yet is still accepted. The final prompt is weaker than iteration-2, but that state is lost.

DEPRO+GATEON (seed 2026). Starting from baseline (acc 0.26), iteration-1 degrades to 0.24 (rolled back), iteration-2 improves to 0.28 (accepted), iteration-3 degrades to 0.27 (rolled back). The best state (0.28) is preserved as final output.

Key difference. Global Rewrite shows higher gains but uncontrolled drift; DEPRO shows controlled progress with explicit rollback protection.

5.6 Discussion

The stability-accuracy tradeoff. Global Rewrite achieves higher peak accuracy but with greater variance (std 0.0511 vs 0.0336 for GATEON). DEPRO trades some peak performance for traceable optimization: every accepted change is tied to one module, every rejected change is explicitly rolled back, and the optimization state remains interpretable at every step. This tradeoff reflects a fundamental tension between search power and process control. Global Rewrite's larger prompt-level jumps enable exploration of distant regions in the prompt space, potentially discovering high-performing configurations that modular refinement cannot reach. However, these jumps are opaque: when a rewrite improves performance, we cannot determine which functional change was

responsible; when it degrades performance, we lose the opportunity to learn from the failure.

Module cooperation and antagonism. Our module ablation reveals cooperation between exact-format and concise-answer modules, and antagonism from evidence-support—findings that holistic rewriting cannot surface. This attribution capability enables systematic identification of compatible module combinations. The evidence-support module's negative interaction suggests that explicit evidence constraints may conflict with the model's internal reasoning process on mathematical problems, where answers are derived through computation rather than extracted from question text. This finding would be invisible under global rewriting, where the evidence-support constraint would be entangled with other prompt modifications.

Implications for prompt engineering practice. DEPRO's modular structure suggests a practical workflow for prompt development: (1) identify distinct behavioral constraints through task analysis, (2) verify orthogonality through ablation experiments, (3) optimize modules independently while monitoring for antagonistic interactions, and (4) freeze or remove modules that consistently degrade performance. This workflow transforms prompt optimization from trial-and-error into a systematic engineering process with clear attribution at each step.

When to use DEPRO versus Global Rewrite. Use Global Rewrite when peak accuracy is the sole objective and variance is acceptable. Use DEPRO when you need to understand why a prompt works, must audit optimization decisions for compliance, or are building reusable prompt libraries. In production settings where changes must be justified, DEPRO's attribution and rollback guarantees may be worth the accuracy cost. For research applications, DEPRO provides a controlled experimental framework for studying prompt component interactions—a capability that global rewriting cannot provide.

6 Conclusion

We presented DEPRO, a modular prompt refinement framework that addresses the functional coupling problem in prompt optimization. By decomposing prompts into named functional modules and restricting each optimization step to modifying one module at a time, DEPRO establishes causal attribution between modifications and performance changes. GATEON, our non-negative acceptance gate, prevents performance degradation and preserves rejected updates as diagnostic evidence.

On GSM8K hard-top300, GATEON(all-modules) improves over ungated modular refinement and achieves a much lower full-budget standard deviation than Global Rewrite (0.0019 vs 0.0184), while preserving interpretable optimization trajectories. Module ablation reveals cooperation between exact-format and concise-answer modules, and antagonism from evidence-support—findings only observable through isolated module editing.

DEPRO does not replace unconstrained search: Global Rewrite remains stronger in absolute accuracy. Its value is instead to make prompt optimization systematic, attributable, and auditable in settings where reliability and traceability matter alongside peak performance. Future work can combine DEPRO's module-level attribution with stronger rewrite operators, adaptive acceptance gates, and automated module discovery.

Limitations

This study is limited by scale and scope. All comparisons use 3–8 seeds, so the results should be read as trend-level evidence rather than definitive statistical claims. The rewrite operator is deliberately simple, which helps isolate the effect of modular scheduling and GATEON but also limits peak accuracy. Our best DEPRO configuration still trails Global Rewrite in absolute accuracy, supporting DEPRO primarily as a stability and attribution framework. Finally, the experiments focus on GSM8K exact-match evaluation and manually defined modules; other domains will require new module definitions and larger-scale validation.

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 *International Conference on Learning Representations (ICLR)*. Accepted as oral presentation.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers

to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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)*, Proceedings of Machine Learning Research, pages 13481–13544. PMLR.

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 *International Conference on Learning Representations (ICLR)*.

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 (ACL)*, pages 8086–8098. Association for Computational Linguistics.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 11048–11064. Association for Computational Linguistics.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2025. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations (ICLR)*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022. Memory-based model editing at scale. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 15817–15831. PMLR.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. GrIPS: Gradient-free, edit-based instruction search for prompting large language models. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 3827–3846. 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 (EMNLP)*, pages 7957–7968. Association for Computational Linguistics.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *International Conference on Learning Representations (ICLR)*.

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

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. 2024. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations (ICLR)*.

Orion Weller, Marc Marone, Nathaniel Weir, Dawn Lawrie, Daniel Khashabi, and Benjamin Van Durme. 2024. "according to ...": Prompting language models improves quoting from pre-training data. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 2288–2301. Association for Computational Linguistics.

Yanzheng Xiang, Hanqi Yan, Lin Gui, and Yulan He. 2024. Addressing order sensitivity of in-context demonstration examples in causal language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6467–6481. Association for Computational Linguistics.

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

Sheng Yang, Yurong Wu, Yan Gao, Zineng Zhou, Bin Benjamin Zhu, Xiaodi Sun, Jian-Guang Lou, Zhiming Ding, Anbang Hu, Yuan Fang, Yunsong Li, Junyan Chen, and Linjun Yang. 2024b. AMPO: Automatic multi-branched prompt optimization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 20267–20279. Association for Computational Linguistics.

Taolin Zhang, Qizhou Chen, Dongyang Li, Chengyu Wang, Xiaofeng He, Longtao Huang, Hui Xue, and

Jun Huang. 2024. DAFNet: Dynamic auxiliary fusion for sequential model editing in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1588–1602. Association for Computational Linguistics.

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 *International Conference on Learning Representations (ICLR)*.

A Appendix Overview

This appendix provides implementation and analysis details that support the main paper. Appendix B gives the exact prompt components used in the experiments and explains how they are serialized into a single model input. Appendix C defines the main experimental objects. Appendix D specifies the optimization procedure and the state invariants maintained by DePRO. Appendix E describes evaluation control, answer matching, and seed handling. Appendix F details the compared methods. Appendix G documents the module-combination protocol. Appendix H describes rejected-update auditing. Appendix I gives the statistical reporting procedure, and Appendix J summarizes the reproducibility configuration.

The appendix is intended to make the experimental protocol auditable. It does not introduce additional experimental claims beyond those reported in the main text. Where the study is underpowered, we preserve the same trend-level interpretation used in the main paper.

B Exact Prompt and Module Definitions

Prompt serialization. Each evaluated prompt is formed by concatenating a fixed base instruction with the currently active functional modules. The base instruction always appears first. Active modules are then appended in a fixed order: evidence-support, exact-format, and concise-answer. Keeping the serialization order fixed prevents module order from becoming a hidden experimental variable.

Module rationale. The three modules correspond to different failure modes observed in mathematical reasoning evaluation. The evidence-support module targets unsupported answer generation. The exact-format module targets serialization errors that make an otherwise correct answer fail exact-match evaluation. The concise-answer module targets overly verbose final responses that obscure the final numeric answer. These modules are intentionally simple because the study isolates the effect of decoupled editing and rollback gating rather than the strength of a complex prompt proposal generator.

Component	Text
Base prompt	You are a precise math solver. Output only the final numeric answer.
evidence-support	Answer must be directly supported by question content.
exact-format	Prefer exact numeric or boolean output.
concise-answer	Avoid verbose rationale in final answer.
Module update rule	Append to selected rule: (be stricter about explicit evidence)

Table 7: Exact strings used in the benchmark implementation.

Boundary convention. A module is treated as a unit of intervention. During a DePRO update, the selected module may be rewritten, but the base prompt and all non-selected modules must remain unchanged. This convention is what makes each accepted or rejected update attributable to one named functional component.

Criterion	Operational interpretation
Semantic independence	The module should express one behavioral constraint rather than a mixture of unrelated instructions.
Replaceability	The module can be edited or removed without changing the base prompt or the syntax of other modules.
Effect attributability	Score changes after an edit can be associated with the edited module because all other components are fixed.

Table 8: Operational interpretation of the three orthogonality criteria used to define prompt modules.

C Notation and Experimental Objects

Table 9 summarizes the notation used across the method, experiments, and appendix.

The retained state P_t^* is the central object in DePRO. It records the best prompt state that survives the acceptance gate. In ungated DePRO, the retained state is simply the most recent candidate. In DePRO+GateON, the retained state is monotonic with respect to the validation score used by the gate.

Symbol	Meaning
P_{base}	Fixed base instruction shared by all modular prompts.
m_k	The k-th functional module.
K	Number of candidate modules. In this study, $K = 3$.
$\mathcal{A}$	Active module set used by a particular configuration.
P_t^*	Retained prompt state after iteration t.
$\tilde{P}_t$	Candidate prompt produced at iteration t.
$s(P)$	Exact-match score obtained by prompt P under the fixed evaluation configuration.
T	Number of optimization iterations.
B	Number of evaluation examples in the current view.

Table 9: Notation used in the DEPRO optimization and evaluation protocol.

D Optimization Procedure and State Invariants

For reproducibility, each DEPRO run follows the same procedure:

1. Construct the initial prompt by concatenating P_{base} with all modules in the active set $\mathcal{A}$.
2. Initialize the retained state P_0^* as the constructed prompt.
3. Select the next active module using round-robin scheduling.
4. Evaluate the retained state P_{t-1}^* under the fixed evaluation configuration.
5. Collect incorrect cases and use them as feedback for the rewrite operator.
6. Rewrite only the selected module, producing candidate prompt $\tilde{P}_t$.
7. Evaluate $\tilde{P}_t$ under the same decoding, examples, and scoring configuration.
8. Apply GATEON: accept the candidate if its score is no worse than the retained score; otherwise roll back and log the rejected update.

The ungated DEPRO baseline follows the same procedure except that every candidate update is accepted. Global Rewrite receives the same feedback signal but rewrites the complete prompt rather than a single module.

State invariants. DEPRO+GATEON maintains three invariants. First, at most one module text is changed per candidate proposal. Second, the retained score cannot decrease after an accepted update because acceptance requires $s(\tilde{P}_t) \geq s(P_{t-1}^*)$. Third, rejected candidates are not discarded silently: each rejection is retained as diagnostic evidence about an unsafe update direction.

Round-robin scheduling. Round-robin scheduling is used to avoid giving one module more proposal opportunities than another. For an active set $\mathcal{A} = \{m_1, m_2, m_3\}$, the update order cycles through the active modules. For $\mathcal{A} = \{m_2, m_3\}$, only exact-format and concise-answer are scheduled. This design keeps optimization budget comparable across module-level configurations.

Logged field	Purpose
Seed	Reconstructs the evaluation order and random choices.
Iteration	Identifies when the proposal was generated.
Active module	Names the only module allowed to change.
Previous module text	Records the pre-edit component.
Candidate module text	Records the proposed replacement or appended clause.
Retained score	Score before applying the candidate.
Candidate score	Score obtained by the candidate prompt.
Decision	Records accept or rollback.
Score delta	Measures the update effect under fixed evaluation conditions.

Table 10: Recommended logging fields for auditing a DEPRO run.

E Evaluation Protocol Details

Evaluation views. The paper reports three views of the same experimental protocol. The primary view uses 100 samples across 8 seeds. The robustness view uses the full hard-top300 subset across 3 seeds. The pilot view uses 100 samples across 3 seeds and is reported as exploratory evidence only. These views trade off sample coverage and seed coverage while keeping the method comparison fixed.

Shared evaluation order. For a given seed and evaluation view, all compared methods are evaluated on the same examples in the same order. This makes method comparisons paired at the seed level and reduces variance caused by different sample draws. The seed does not change the underlying

task, gold answer, prompt template, model endpoint, or decoding parameters.

Exact-match objective. The objective is exact-match accuracy on the final answer. A prediction is counted as correct only when the evaluated final answer matches the gold answer under the same benchmark implementation used for all methods. We do not apply method-specific post-processing or method-specific answer cleaning. This is important because the exact-format and concise-answer modules are intended to affect the model behavior itself rather than a downstream parser.

Why trend-level reporting. The reported seed counts are intentionally described as trend-level evidence. Small seed counts can identify consistent directionality, large effect sizes, and obvious instability, but they should not be interpreted as definitive statistical proof. The paper therefore emphasizes effect magnitude, consistency across evaluation views, and qualitative interpretability.

F Compared Method Implementations

Manual. The manual baseline uses only the fixed base prompt and performs no optimization. It establishes the performance level of the human-written instruction before any automated search.

Global Rewrite. Global Rewrite treats the prompt as one monolithic text block. At each iteration, the proposal generator receives the current prompt and feedback from incorrect cases, then produces a complete replacement prompt. Because many prompt functions can change in one step, score changes cannot be attributed to a single functional component.

DePRO baseline. The ungated DePRO baseline uses the same modular decomposition and round-robin scheduling as DePRO+GateON, but it accepts every candidate update. This isolates the contribution of the rollback gate: any difference between ungated DePRO and DePRO+GateON is due to acceptance control, not to module decomposition or scheduling.

DePRO+GateON. DePRO+GateON uses modular proposals and accepts only non-degrading candidates. Its retained state therefore reflects the best accepted trajectory under the validation score used by the gate. The method is designed for attribution and stability rather than unconstrained search power.

Method	Unit edited	Gate	Main diagnostic value
Manual	None	None	Fixed reference point
Global Rewrite	Whole prompt	None	Peak-search comparison
DePRO baseline	One module	None	Effect of modularization alone
DePRO+GateON	One module	Non-negative	Stability and rollback attribution

Table 11: Implementation distinction among compared methods.

G Module Combination Protocol

Active set	evidence	exact	concise
all-modules	✓	✓	✓
exact-only		✓	
exact+concise		✓	✓
exact+concise+evidence	✓	✓	✓

Table 12: Module combinations used to inspect cooperation and antagonism. The last row repeats the full set in the pilot naming convention.

Interpretation of module interactions. Module combinations are used to inspect whether two functional constraints cooperate, interfere, or behave approximately independently. Cooperation is suggested when the combined active set outperforms the corresponding individual or larger set under the same evaluation view. Antagonism is suggested when adding a module consistently reduces accuracy or increases instability. These interpretations are treated as exploratory unless they are stable across evaluation views.

Why exact-format and concise-answer are paired. The exact-format and concise-answer modules target related but separable failure modes. Exact-format asks for the answer to be expressed in the expected answer type, while concise-answer discourages additional rationale in the final output. Pairing them tests whether output serialization and answer brevity provide complementary benefits under exact-match evaluation.

Why evidence-support is inspected separately. The evidence-support module is useful for testing whether grounding-oriented constraints help mathematical reasoning prompts. In the reported pilot view, it exhibits negative interaction with the exact+concise pair. The main paper treats this as a

module-interaction finding rather than a universal claim that evidence constraints are harmful in all tasks.

H Rejected Update Categories

Manual inspection of rejected updates suggests three recurring categories:

- **Over-specification**: the update adds overly strict requirements that reduce answer coverage.
- **Directive conflict**: the update introduces a rule that competes with another active module, such as asking for detailed reasoning while the concise-answer module demands brevity.
- **Formatting brittleness**: the update imposes an output schema that the model does not reliably follow under exact-match evaluation.

These categories are not used as training labels; they are diagnostic summaries of why GATEON rejected harmful candidate prompts.

Use of rejected updates. Rejected updates are valuable because they preserve negative evidence. In a standard whole-prompt rewrite system, a failed candidate is typically discarded after evaluation. In DEPRO, the failed candidate is linked to one active module, making it possible to ask which functional direction was unsafe. This audit trail supports post-hoc analysis of module compatibility.

Audit interpretation. A rejection does not imply that the selected module is globally harmful. It only shows that the proposed edit was harmful under the current prompt state, evaluation set, and decoding configuration. A later proposal to the same module may still be accepted if it satisfies the non-negative gate.

Category	Typical symptom
Over-specification	The model becomes too constrained and fails to answer some otherwise answerable questions.
Directive conflict	The new instruction competes with another active module, producing unstable behavior.
Formatting brittleness	The model is asked to emit a structure that is fragile under exact-match scoring.

Table 13: Qualitative categories used when inspecting rejected updates.

I Statistical Significance Details

Seed-level paired deltas. For each comparison, we compute a paired delta at the seed level:

$$\Delta_i = s_i(\text{method A}) - s_i(\text{method B}), \quad (3)$$

where i indexes a shared seed. The reported mean delta is

$$\bar{\Delta} = \frac{1}{n}\sum_{i=1}^{n} \Delta_i. \quad (4)$$

Pairing by seed reduces variation from sample order and seed-specific effects.

Cross-seed standard deviation. For a method with seed-level scores $s_1, \ldots, s_n$, the reported standard deviation is

$$\mathrm{std}(s) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(s_i - \bar{s})^2}. \quad (5)$$

Because n is small, standard deviation should be read as a descriptive measure of stability rather than a precise estimate of the population variance.

Bootstrap confidence intervals. Bootstrap confidence intervals are computed over seed-level paired deltas. The resampling unit is the seed, not an individual example, because methods are compared under shared seeds. With only 3–8 seeds, the interval should be read as an uncertainty summary rather than a strong significance claim.

Sign test. The sign test counts how often one method has a positive paired delta over another across seeds. It does not assume normality, but it is conservative under small seed counts. We therefore report sign-test values for transparency while avoiding strong claims of statistical significance.

J Reproducibility Configuration

Runtime controls. All methods share the same model endpoint, temperature, maximum output length, timeout, sample budget, and evaluation objective. These controls are necessary because the purpose of the experiment is to compare optimization procedures, not different inference budgets.

Deterministic decoding. Temperature is set to 0.0 to reduce sampling noise. This setting does not remove all possible runtime variation from model serving infrastructure, but it keeps decoding as controlled as possible under the experimental setup.

View	Comparison	Δ mean	95% CI	Sign-p
100×8	GATEON(all) vs. baseline	$+0.0150$	$[+0.0025, +0.0325]$	0.125
100×8	e+c vs. GATEON(all)	-0.0063	$[-0.0213, +0.0087]$	0.891
full300×3	GATEON(all) vs. baseline	$+0.0067$	$[+0.0000, +0.0100]$	0.250
full300×3	e+c vs. GATEON(all)	-0.0033	$[-0.0100, +0.0000]$	1.000

Table 14: Seed-level paired deltas and bootstrap confidence intervals. Results are trend-level evidence given the small seed count.

Field	Value
Task	GSM8K hard-top300
Model	Qwen3-4B-Thinking-2507
Objective	Exact-match accuracy
Samples per seed	100 (primary) / 300 (robustness)
Seeds	2026–2033 (primary); 2026–2028 (full top300)
Iterations	3
Temperature	0.0
Max output tokens	512
Timeout	60 sec

Table 15: Reproducibility configuration.

Artifact checklist. Table 16 lists the artifacts needed to reproduce the reported results. The paper's claims rely on keeping these artifacts fixed across methods.

Artifact	Role in reproduction
Hard-top300 index list	Fixes the evaluation instances.
Prompt component file	Records the base prompt and module strings.
Seed list	Reconstructs primary and robustness evaluation views.
Method configuration	Specifies active modules, gate setting, and iterations.
Run logs	Records accepted and rejected updates.
Prediction file	Stores model outputs before scoring.
Score script	Computes exact-match accuracy under one shared rule.

Table 16: Recommended artifact checklist for reproducing the experiment.

K Additional Notes on Scope

Why the appendix is descriptive. The appendix expands protocol details but does not add new claims about model performance. This distinction is important: the main paper contains the quantitative results, while the appendix clarifies how those results are constructed, interpreted, and audited.

Generalization beyond GSM8K. The same modular framework can be applied to other tasks only after redefining task-appropriate modules. For dialogue, modules might represent safety, helpfulness, and persona constraints. For code generation, modules might represent test-passing behavior, API constraints, and formatting. Such extensions require new orthogonality checks because module interactions are task-specific.

Practical recommendation. When applying DEPRO in a new domain, the recommended workflow is: identify common failure modes, convert each failure mode into a candidate module, verify replaceability by removing or rewriting each module, run a small ablation study, and only then use GATEON for longer optimization. This keeps module discovery separate from module refinement and makes the resulting prompt library easier to audit.