DIFFERENTIAL PROMPT OPTIMIZATION: PINPOINTING STUDENT ERRORS VIA TEACHER AUDITS

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic prompt optimization typically guides instruction search using holistic evaluations or end-to-end task success. For complex multi-step reasoning, however, attributing a failure across a long trajectory remains difficult, as an argument may depart from a viable path long before its final line. Without fine-grained diagnostics, the optimizer must infer the cause of failure from sparse signals alone. We introduce Differential Prompt Optimization (DIFFPO), a framework that supplies step-level diagnostic credit in natural language. During training, an auxiliary teacher model conducts a differential audit on failed student traces: it pinpoints the earliest step of divergence where the student's reasoning becomes unviable and abstracts that error into a generalizable rule for the prompt optimizer. To support the student at test time, the teacher also builds a library of general problem-solving strategies. The student selectively retrieves these strategies to guide its reasoning, providing high-level structure. Across different reasoning and instruction-following benchmarks, DIFFPO consistently outperforms relevant prompt optimization baselines.

1 INTRODUCTION

Automatic prompt optimization treats prompt engineering as a discrete search problem. Typically, an optimizer model proposes candidate instructions, evaluates them against a validation batch executed by a task model, and iteratively refines the prompt using execution feedback (Zhou et al., 2023; Yang et al., 2024). Substantial progress has been made in strengthening the search mechanism, including gradient-free editing (Prasad et al., 2023), Monte Carlo tree search (Wang et al., 2024), and evolutionary algorithms (Fernando et al., 2024; Guo et al., 2024). Across these search paradigms, however, the feedback signal supplied to the optimizer remains fundamentally sparse: an instance-level score indicating whether the final answer was correct.

While coarse outcome feedback may suffice for frontier models capable of autonomous self-reflection, it creates a bottleneck in models commonly used in production environments. Deployed systems rely on compact language models, which have less representational capacity to recover from ambiguous instructions and are prone to intermediate reasoning breakdowns.

When a model fails on a multi-step task, attributing the final failure to a misread constraint, an unviable strategy, or a clerical slip in the final step becomes an intractable credit assignment problem. In reinforcement learning, this challenge motivated the transition from outcome supervision to process supervision, validating solutions step-by-step (Lightman et al., 2024; Uesato et al., 2022). In prompt optimization, however, step-level verifiers and reward gradients are unavailable. Although recent prompt distillation approaches attempt to transfer reasoning logic from larger models to smaller ones (Boateng et al., 2025; Badhe & Shah, 2026), they typically compile static, instance-agnostic guidelines in advance, failing to account for the specific failure modes of the downstream model.

We introduce Differential Prompt Optimization (DIFFPO), an optimization framework that provides step-level diagnostic supervision in natural language. DIFFPO decouples task execution from diagnosis by bringing in a teacher model during training. Whenever the student fails to solve a problem, the teacher conducts a *differential audit*: it inspects the student's reasoning trace alongside the reference to find where the student failed. Because problems may have multiple valid solution paths, the teacher follows along with the student's chosen approach and flags the step where that reasoning becomes unrecoverable. It then translates the mistake into a general rule rather than a problem-specific

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

patch. This gives the prompt optimizer targeted, actionable guidance that addresses the student's actual blind spots without overfitting to individual training examples.

Furthermore, smaller models often benefit from seeing how a related problem is structured. However, supplying standard in-context examples may introduce practical problems may cause the smaller models to blindly copy numbers and variables from the demonstration into the target problem. To provide procedural guidance without these drawbacks, DIFFPO pairs the prompt audit with a *strategy index*. In summary with the help of the teacher, we frame examples into abstract strategies that help the student without further confusing it.

In summary, our key contributions are:

- **Differential Audit Framework:** We introduce DIFFPO, decoupling execution from diagnosis by using a teacher to pinpoint logical divergence point and translate it into a transferable rule.
- **Hierarchical Strategy Scaffolding:** We formulate abstract procedural indexing to inject help the student with similar examples.
- **Empirical Validation:** Across three diverse domains (**MATH-`Hard`**, **CodeIO-`Puzzle`**, and **IF-Bench**) and two model families (Gemini and Gemma), DIFFPO consistently outperforms state-of-the-art reflective optimizers and prompt distillation baselines.

2 RELATED WORK

Automatic prompt optimization. Automatic Prompt Engineer (APE) (Zhou et al., 2023) and OPRO (Yang et al., 2024) formalized prompt optimization as a discrete search problem: an LLM optimizer proposes candidate instructions, evaluates them on a training batch using a task model, and uses the resulting scores to guide subsequent proposals. Subsequent work has primarily focused on strengthening this search space exploration (Prasad et al., 2023; Wang et al., 2024; Fernando et al., 2024; Guo et al., 2024; Kumar et al., 2025; Sharma & Henley, 2026; Zhou et al., 2025).

Reflective search and failure diagnosis. To provide richer updates, recent reflective optimization methods feed individual execution failures directly to an optimizer LLM. TextGrad (Yuksekgonul et al., 2024), GEPA (Agrawal et al., 2025), and APEX (Wang et al., 2026) evaluate execution traces as textual gradients. These frameworks share the input problem, the model's generated output, and a final correctness label with the reflection model (optimizer). The optimizer is then left to infer on its own whether a failure stemmed from a misread premise.

Dynamic context and exemplar abstraction. Complementing instruction optimization, a separate line of research provides procedural scaffolding at inference time. Moving beyond static in-context demonstrations (Dong et al., 2024; Min et al., 2022), recent frameworks (Zhang et al., 2026; Pei et al., 2025) treat context as an evolving operational playbook, while Suzgun et al. (2026); Liang et al. (2026) maintain searchable repositories of problem-solving insights.

Knowledge distillation in prompt optimization. Frameworks such as Prompt-Level Distillation (Badhe & Shah, 2026) and Concept Distillation (Boateng et al., 2025) prompt a larger teacher to extract high-level rules from a dataset and compile them into a static system prompt for a smaller student. Other approaches train auxiliary advisor models to rewrite instructions for frozen downstream models (Cheng et al., 2024; Asawa et al., 2025), or align multi-scale models via reinforcement learning (Lu et al., 2026; Liu et al., 2026). In existing distillation methods, the teacher's guidance is compiled statically in advance, reflecting what the teacher assumes is important rather than the specific, empirical reasoning bottlenecks of the student.

Positioning DIFFPO. DIFFPO combines the teacher–student division of labor from prompt distillation with reflective prompt search, while differing from both along two primary axes. First, the teacher diagnoses the student's trace to locate the incorrect step and formulate that breakdown as a transferable rule. Second, rather than relying on prompt instructions alone, DIFFPO provides teacher-distilled exemplars to further improve student performance.

3 METHODOLOGY

We introduce DIFFPO, an optimization framework designed to improve prompt quality for compact language models. Standard reflective prompt optimization relies on a setup where an optimizer

model diagnoses multi-step reasoning breakdowns from coarse terminal outcomes alone. DIFFPO departs from this paradigm by adopting a teacher–student design: we delegate error diagnosis and procedural synthesis to a high-capacity teacher model during optimization. The primary component of our framework is the **Differential Audit** (§3.5): the teacher inspects the student's step-by-step reasoning trace against a reference solution, localizes the point of logical divergence, and abstracts the failure into a diagnostic rule for prompt refinement.

Furthermore, in several prior works, the optimizer accumulates all failure feedback acquired during training directly into the system prompt. However, static guidelines are rarely universally applicable across an entire dataset, while omitting procedural guidance entirely deprives the downstream model of necessary execution support. To resolve this trade-off, we propose **Abstract Strategy Indexing** (§3.2–§3.4). When sufficient auxiliary problem data is available, this optional component compiles a pool of solved problems into hierarchical procedural blueprints, equipping the student with dynamic structural guidance at inference time.

3.1 PROBLEM FORMULATION

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ denote the training dataset, where $x_i \in \mathcal{X}$ is a problem statement and $y_i \in \mathcal{Y}$ is its reference answer. Given an instructional prompt $\pi \in \Omega$ and a problem x, a compact student model $\mathcal{M}_S$ generates a multi-step reasoning trace $\hat{s}$ alongside a final answer prediction $\hat{y}$:

$$\big(\hat{s}, \hat{y}\big) \sim \mathcal{M}_S(\,\cdot \mid x; \pi). \tag{1}$$

The objective of automatic prompt optimization is to discover an instruction $\pi^* \in \Omega$ that maximizes expected downstream task utility:

$$\pi^* = \arg\max_{\pi \in \Omega} \; \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\,\mathcal{U}(\hat{y}, y)\,\big], \tag{2}$$

where $\mathcal{U}\colon \mathcal{Y} \times \mathcal{Y} \to \{0, 1\}$ evaluates final answer correctness against the reference label.

In reflective prompt optimization frameworks (Yuksekgonul et al., 2024; Agrawal et al., 2025; Wang et al., 2026), an optimizer language model iteratively updates π by analyzing execution failures over a batch $\mathcal{B} \subset \mathcal{D}$. For an incorrect instance i, the optimizer inspects a standard observation tuple:

$$\text{feedback}_i = \big(x_i,\ \hat{s}_i,\ \hat{y}_i,\ \mathcal{U}(\hat{y}_i, y_i)\big). \tag{3}$$

Because feedback_i contains only terminal outcome feedback, attributing failure across a multi-step trace $\hat{s}_i$ presents a challenging problem. The optimizer is left to infer whether the error originated from a misread constraint, an unviable heuristic, or a clerical calculation slip. When applied to compact models with subtle reasoning bottlenecks, this lack of intermediate supervision yields noisy prompt updates. DIFFPO addresses this challenge by providing localized diagnostic supervision through differential audits during training (§3.5).

3.2 OFFLINE PHASE: STRATEGY INDEX CONSTRUCTION

In-context demonstrations enable language models to infer task conventions and improve execution consistency (Dong et al., 2024; Min et al., 2022). However, prepending conventional problem–solution pairs to compact models introduces two key challenges: (i) standard dense retrievers over raw problem text predominantly match on superficial lexical overlap (e.g., shared entity names) rather than underlying algorithmic structure; and (ii) compact models frequently exhibit severe exemplar copy-bias, directly importing arbitrary variables, numerical constants, or intermediate states from the demonstration into their own reasoning traces.

To resolve both bottlenecks, DIFFPO decouples offline procedural synthesis from online execution, compiling an index of verified reference problems paired with consolidated procedural abstractions.

We initialize the strategy knowledge base $\mathcal{KB}$ from an auxiliary problem pool $\mathcal{D}_{\text{pool}} = \{(x_j, y_j)\}_{j=1}^{M}$, constructed to be strictly disjoint from both the prompt optimization batch $\mathcal{B} \subset \mathcal{D}$ and the test split. To obtain verified step-by-step reasoning traces, the teacher model $\mathcal{M}_T$ synthesize a candidate solution for each problem. However, injecting verified solutions directly into prompts may cause constant bleed, whereas omitting concrete solutions entirely deprives compact students of necessary procedural guidance. We balance this trade-off by extracting a blueprint that frames

the concrete solution of data points. For each problem (x_j, y_j), we prompt $\mathcal{M}_T$ to generate both the solution and its structural abstraction:

$$\mathcal{M}_T(x_j,\, y_j) = (s_j,\, \tau_j), \tag{4}$$

where τ_j summarizes the domain context (problem taxonomy), the formulation insight, and the core procedural strategy (an algorithmic outline). We retain an instance only if the teacher solution s_j terminates in the ground-truth target y_j, filtering out flawed intermediate trajectories to prevent propagating teacher errors. Each verified index entry is stored as:

$$k_j = (x_j,\, \tau_j,\, s_j,\, y_j) \in \mathcal{KB}. \tag{5}$$

At retrieval time, candidate entries are matched using their distilled abstractions τ_j (§3.3).

Computational overhead and modularity. Reference generation and abstraction run strictly once offline, requiring one teacher call per retained entry. At test time, retrieval reduces to a standard dense lookup, introducing zero additional teacher calls and minimal serving latency. Importantly, this offline indexing is strictly non-recurring: test-time inference relies entirely on local vector matching, preserving the low latency and zero parameter-overhead required for compact model deployment. Furthermore, when an auxiliary problem pool $\mathcal{D}_{\text{pool}}$ is unavailable, the retrieval component can be omitted, gracefully reducing DIFFPO strictly to the differential prompt audit (§3.5).

3.3 THRESHOLD-GATED STRATEGY RETRIEVAL

At test time, given a target problem x, the student retrieves relevant procedural guidance from $\mathcal{KB}$. To align queries based on latent mathematical logic rather than surface phrasing, a dense encoder E embeds the problem statement and the abstraction τ_j. Relevance is evaluated via cosine similarity:

$$\mathrm{Sim}(x, j) = \frac{\langle E(x),\, E(\tau_j)\rangle}{\|E(x)\|\,\|E(\tau_j)\|}. \tag{6}$$

We deliberately index the abstraction rather than the reference problem text x_j, as raw problem text carries surface entities and numbers that dominate lexical similarity.

Because a finite knowledge base cannot guarantee coverage for every query, unconstrained nearest-neighbor retrieval risks injecting irrelevant exemplars into the student's context. To prevent negative transfer on out-of-distribution queries, we gate candidate admission with a similarity threshold γ:

$$\mathcal{N}_K(x) = \Big\{ j \in \text{Top-}K_{j' \in \{1,\ldots,|\mathcal{KB}|\}} \big(\mathrm{Sim}(x, j')\big) \;\Big|\; \mathrm{Sim}(x, j) \geq \gamma \Big\}. \tag{7}$$

When no candidate satisfies the threshold ($\mathcal{N}_K(x) = \emptyset$), the retrieval mechanism abstains, and the student answers conditioned on the instruction π alone. This retrieval procedure is identical during prompt optimization and evaluation, ensuring instructions are tuned under deployable conditions.

3.4 EXEMPLAR REPRESENTATION TAXONOMY

Retrieving an analogous entry from $\mathcal{KB}$ leaves open how much of its content to expose to the student model. Exposing concrete solutions encourages copy-bias, whereas providing only abstract labels withholds the operational steps required to execute multi-step calculations. We treat this trade-off as a core architectural dimension and formalize three representation formats as follows. For each retrieved entry $j \in \mathcal{N}_K(x)$, the context manager formats the guidance into a structured prompt block prepended to the target problem x. We evaluate the representation variants to systematically decouple *procedural abstraction* (τ_j) from *concrete execution* (x_j, s_j, y_j):

- Φ_{full} supplies the concrete demonstration verbatim (standard few-shot prompting).
- Φ_{micro} isolates the constant-free blueprint. Because τ_j excludes instance-specific variables, the student is exposed to no extraneous numerical values (zero-bleed setting).
- Φ_{insight} deploys full hierarchical scaffolding. Surfacing the formulation invariant τ_j acts as an attentional anchor, encouraging the student to treat the solution s_j as an algorithmic instance rather than a source of literal constants to copy.

3.5 THE DIFFERENTIAL AUDIT

The differential audit supplies the diagnostic attribution that terminal outcome feedback equation 3 omits. When the student answers a training problem incorrectly, the teacher identifies where the reasoning broke down and abstracts the failure into an actionable rule.

Offline reference caching. We precompute and cache a verified reference solution $s_i \sim \mathcal{M}_T(\cdot \mid x_i)$ for every problem in the training set $\mathcal{D}$ (provided the teacher can solve it). This caching ensures reference generation requires only a single teacher call per instance.

Fault diagnosis and rule extraction. When the student produces an incorrect response $(\hat{s}_i, \hat{y}_i)$ with $\mathcal{U}(\hat{y}_i, y_i) = 0$, the teacher inspects the problem x_i, the reference s_i, and the student's trace $\hat{s}_i$:

$$a_i = \mathcal{M}_{T\text{audit}}(x_i, s_i, \hat{s}_i) = (d_i, r_i). \tag{8}$$

Here, the diagnosis d_i pinpoints where the student's reasoning in $\hat{s}_i$ broke down and explains the error, while r_i abstracts that breakdown into an instance-invariant rule stripped of problem-specific quantities. The feedback passed to the optimizer is enriched with this diagnosis:

$$\text{feedback}_i^{\text{diff}} = \big(x_i,\ \hat{s}_i,\ \hat{y}_i,\ \mathbf{a}_i\big). \tag{9}$$

Because correct student responses bypass auditing, diagnostic compute scales directly with the student's empirical failure rate.

Audit design principles. Two design principles govern the audit prompt:

Path tolerance: The teacher is instructed to focus on the student's output rather than the solution s_i.

Rule abstraction: The teacher must formulate r_i as a generalizable heuristic that provides actionable guidance that transfers across tasks.

Batch aggregation and prompt refinement. At the end of an evaluation batch $\mathcal{B}$, failure audits $\{\mathbf{a}_i\}$ are grouped by rule r_i. The unique rules, ranked by empirical frequency, are supplied to the optimizer OPT alongside the current instruction and its batch accuracy. The optimizer proposes revised instructions targeting these common failure modes. Candidate instructions are evaluated on the batch, and the top candidates are retained for the next iteration.

In summary, rather than aggregating raw batch failures and relying on the optimizer to simultaneously diagnose errors and propose revisions, DIFFPO introduces a teacher model into the training loop to perform fine-grained, instance-level diagnosis. Furthermore, when an auxiliary knowledge base is available, the teacher provides structured execution guidance by supplying abstract blueprints and exemplars.

4 EXPERIMENTS

4.1 SETUP

Datasets. We evaluate DIFFPO across three distinct reasoning domains. For mathematical reasoning, we train on 500 Level-5 problems from MATH (Hendrycks et al., 2021) and evaluate on all Level-5 problems in **MATH500**; we denote this benchmark split as **MATH-`Hard`**. For logical reasoning, we use **CodeIO-`Puzzle`**, we train on a randomly selected subset of 1000 problems from **CodeIO** (Li et al., 2025) designated for *Modeling of Logical Puzzles and Complex Reasoning Processes* and evaluate on all the test data in the same category. For instruction following, we randomly select 500 instances from **IFBench** (Pyatkin et al., 2026) and evaluate on the entire test set.

Model configurations. Our evaluation involves the three functional roles formalized in §3: the compact student model $\mathcal{M}_S$, the prompt optimizer OPT, and the teacher model $\mathcal{M}_T$ (for methods that lack a teacher, only $\mathcal{M}_S$ and OPT are active). To ensure strict parity, all baselines and DIFFPO share identical model assignments across roles under two regimes:

1) **Gemini:** $\mathcal{M}_S$ is instantiated as `gemini-3.1-flash-lite` , while both OPT and $\mathcal{M}_T$ are `gemini-3-flash`. For models with configurable reasoning depth, the thinking budget is set to `HIGH` for OPT and $\mathcal{M}_T$, `MINIMAL` for $\mathcal{M}_S$ on **MATH-`Hard`** and **IFBench**, and `LOW` for $\mathcal{M}_S$ on **CodeIO-`Puzzle`**. 2) **Gemma:** $\mathcal{M}_S$ is a `gemma-v4-it-12b`, with `gemma-4-31b-it` serving as OPT and $\mathcal{M}_T$.

Both configurations query models through standard inference interfaces without gradient or weight access. In all experiments, the student model operates with temperature 0, whereas the optimizer and teacher temperatures are set to 1.0.

Baselines. We compare against state-of-the-art reflective prompt optimizers and teacher-distillation baselines: 1) **Initial prompt:** The unoptimized seed instruction shared across all methods. 2) **GEPA** (Agrawal et al., 2025): Evolves candidate instructions through reflective mutation across an empirical Pareto frontier. 3) **APEX** (Wang et al., 2026): A sample-efficient reflective optimizer that stratifies the training set to concentrate evaluation on informative failure cases. 4) **PLD** (Badhe & Shah, 2026) and **CD** (Boateng et al., 2025): Teacher-guided baselines that compile offline diagnostic rules or conceptual guidelines into a static system prompt. Finally, because DIFFPO is agnostic to the underlying algorithm, we integrate it on top of both reflective optimizers, denoted **DIFF-GEPA** and **DIFF-APEX** .

To account for optimization stochasticity, all experiments are repeated across 5 distinct random seeds. For each method, we select the top-3 candidate instructions ranked by their training objective score $\mathcal{S}$ and report their mean test accuracy alongside standard deviations. To account for teacher query overhead, we scale the number of optimization iterations for all baselines to match the total LLM query budget of DIFFPO (excluding the knowledge base creation). To retrieve exemplars from the knowledge base, we set the $\gamma = 0.5$ and $K = 3$ for **IFBench** and **MATH-`Hard`** and $K = 1$ for **CodeIO-`Puzzle`**. Full implementation details and hyperparameter configurations are provided in Appendix C.

4.2 MAIN RESULTS

Tables 1 and 2 report test accuracy across the Gemini and Gemma model suites. Across both model families and all three reasoning benchmarks, DIFFPO consistently improves downstream performance over the baseline methods. Three primary empirical trends emerge from these results:

Consistent gains across search backbones. Integrating DIFFPO improves the final accuracy of both GEPA and APEX across all benchmarks. As DIFFPO operates as an orthogonal diagnostic and representation scaffold, it delivers cumulative improvements. In particular, on **IFBench**, where candidate instructions must satisfy complex compositional constraints, DIFFPO drives substantial improvements over vanilla reflective baselines. Similarly, on **CodeIO-`Puzzle`**, where baseline methods fail to breach empirical plateaus, both DIFF-GEPA and DIFF-APEX achieve notable step-changes.

Cross-architecture generalization. The Gemma evaluations (Table 2) replicate the gains observed on Gemini models under similar hyperparameter settings. This confirms that the framework's diagnostic attribution and abstract retrieval mechanisms do not rely on capabilities of frontier models.

Method	MATH-`Hard`	CodeIO-`Puzzle`	IFBench	Average
Initial prompt	83.58	57.33	44.00	61.64
GEPA (Agrawal et al., 2025)	85.08 ± 1.29	58.16 ± 0.29	61.56 ± 2.91	68.27
APEX (Wang et al., 2026)	85.77 ± 0.87	58.11 ± 0.77	67.92 ± 3.76	70.60
PLD (Badhe & Shah, 2026)	84.57 ± 0.86	57.89 ± 0.38	60.89 ± 2.13	67.78
CD (Boateng et al., 2025)	86.32 ± 1.88	58.39 ± 0.26	55.70 ± 2.55	66.80
DIFF-GEPA (**Ours**)	86.07 ± 0.43	$\mathbf{60.94 \pm 0.17}$	73.92 ± 1.09	73.64
DIFF-APEX (**Ours**)	$\mathbf{86.57 \pm 1.50}$	60.11 ± 0.51	$\mathbf{74.94 \pm 1.19}$	**73.87**

Table 1: Downstream benchmark accuracy using Gemini models.

4.3 ANALYSIS

4.3.1 IMPACT OF EXEMPLAR REPRESENTATION FORMATS

Table 3 compares the representation variants defined in §3.4 under identical optimization configurations across **IFBench** and **CodeIO-`Puzzle`**. The empirical results confirm the structural trade-off motivated in our formulation:

Method	MATH-Hard	CodeIO-Puzzle	IFBench	Average
Initial prompt	82.84	53.17	41.16	59.06
GEPA (Agrawal et al., 2025)	86.32 ± 0.86	52.56 ± 0.20	49.66 ± 0.34	62.85
APEX (Wang et al., 2026)	85.32 ± 0.86	53.17 ± 0.60	56.06 ± 2.51	64.85
PLD (Badhe & Shah, 2026)	85.57 ± 0.43	53.17 ± 0.34	46.49 ± 4.57	61.74
CD (Boateng et al., 2025)	87.07 ± 1.56	53.33 ± 0.88	45.15 ± 1.01	61.85
DIFF-GEPA (**Ours**)	87.56 ± 0.43	57.11 ± 0.51	50.91 ± 3.09	65.19
DIFF-APEX (**Ours**)	$\mathbf{88.31 \pm 1.14}$	$\mathbf{57.16 \pm 0.58}$	$\mathbf{64.40 \pm 3.79}$	**69.96**

Table 2: Downstream benchmark accuracy using Gemma models.

Dataset	Method	Φ_{insight}	Φ_{full}	Φ_{micro}
IFBench	DIFF-GEPA	$\mathbf{73.92 \pm 1.09}$	64.47 ± 1.38	65.36 ± 0.86
	DIFF-APEX	$\mathbf{74.94 \pm 1.19}$	69.16 ± 3.35	73.70 ± 0.39
CodeIO-Puzzle	DIFF-GEPA	$\mathbf{60.94 \pm 0.17}$	59.45 ± 0.39	58.61 ± 1.25
	DIFF-APEX	$\mathbf{60.11 \pm 0.51}$	59.11 ± 0.10	58.78 ± 0.19

Table 3: Performance of different exemplar representations on **CodeIO-Puzzle** and **IFBench** using `gemini-3.1-flash-lite`. The results confirm that well-structured dynamic context is critical to achieving superior performance.

A critical aspect of dynamic context design is how to present guidance effectively, particularly for compact language models. Our investigation, on **IFBench** and **CodeIO**, shows that pairing the formulation blueprint τ_j with the concrete solution s_j consistently yields the highest accuracy. Including τ_i acts as an attentional anchor, allowing the student to use s_j for procedural grounding. As Table 3 shows, providing the strategy alongside the question–solution demonstration provides the greatest performance benefit.

4.3.2 COMPONENT ABLATIONS AND DESIGN CONTROLS

To isolate the source of empirical improvements and evaluate our compute allocation, Table 4 analyzes individual component removals alongside two compute-matched control baselines.

Differential teacher audit (w/o Teacher). Removing the step-level differential audit and reverting to standard feedback causes the steepest performance collapse across both search algorithms, sharply reducing accuracy. This confirms that even with proper examples in the context, coarse credit assignment leaves the optimizer unable to isolate the root cause of multi-step reasoning failures, yielding unstable prompt updates.

Strategy blueprint retrieval (w/o Exemplars). Omitting retrieval from $\mathcal{KB}$ forces the student to rely strictly on the optimized prompt π^* without dynamic procedural blueprints. While performance remains solid (substantially outperforming baselines), it consistently trails the full framework. This indicates that while the differential audit optimizes strong foundational instructions, dynamic procedural scaffolding provides essential execution grounding on challenging tail instances.

Naive scaffolding (+ Φ_insight). A natural question is whether DIFFPO's improvements stem solely from exemplar retrieval rather than prompt optimization. To test this, we prepend retrieved Φ_{insight} blueprints to final prompts produced by standard baselines (GEPA and APEX) without differential auditing during training. Crucially, naive blueprint injection yields negligible gains on APEX (70.07% vs. 74.94%) and induces an outright performance degradation on GEPA (60.43% vs. 61.56%). Because compact models are prone to distraction, instructions must be explicitly optimized in the presence of hierarchical scaffolding to teach the student how to ground reasoning in abstract blueprints without copying exemplar artifacts.

Compute-matched teacher-only optimization (Teacher Only). We also address the counterhypothesis: if compute is available to construct an auxiliary knowledge base, should that budget instead be spent running standard prompt optimization directly on a teacher model? We optimize prompts using `gemini-3-flash` scaled to match the total query budget of our offline indexing phase, and deploy the resulting prompt to the compact student. As shown in Table 4, this strategy

	DIFFPO Variants			Comparisons & Controls	
Search Backbone	**Full**	**w/o Exemplars**	**w/o Teacher**	**+ Φ_{insight}**	**Teacher Only**
APEX	$\mathbf{74.94 \pm 1.19}$	73.81 ± 0.34	69.84 ± 5.10	70.07 ± 3.37	58.62 ± 5.02
GEPA	$\mathbf{73.92 \pm 1.09}$	72.22 ± 2.75	62.36 ± 1.10	60.43 ± 1.19	55.89 ± 0.86

Table 4: **Component ablation and diagnostic controls on IFBench.** *Full* denotes DIFFPO under Φ_{insight}. *w/o Exemplars* omits retrieval; *w/o Teacher* replaces the differential audit with terminal outcome feedback. + $\Phi_{insight}$ appends retrieved blueprints to the best prompt found via the baseline.

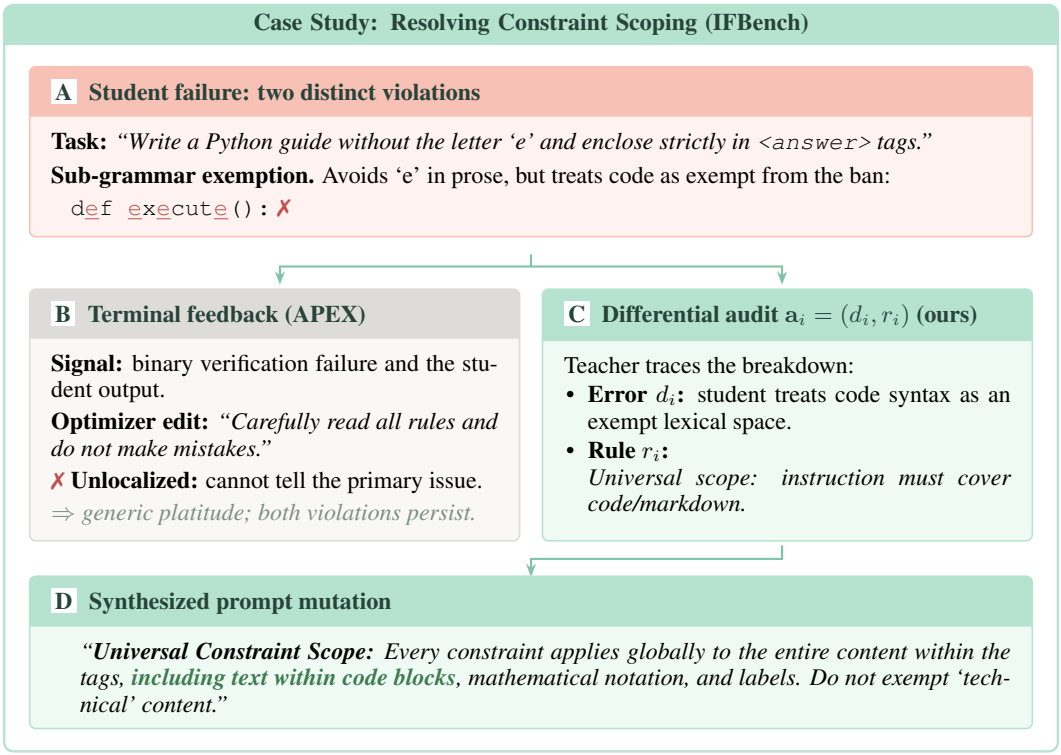

Figure 1: **Differential Audit flow on IFBench dataset**.

fails catastrophically (achieving only 58.62% on APEX and 55.89% on GEPA), well below the unoptimized baseline. Because frontier models possess distinct latent capabilities and reasoning priors, instructions tuned to teacher failure modes fail to address the specific representational bottlenecks of compact students. Effective prompt optimization requires a student-in-the-loop search guided by teacher diagnosis.

Teacher Audit example Figure 1 illustrates a representative audit during optimization. By analyzing the question and the student's breakdown directly, the teacher isolates the primary source of failure; instead of passing an unmanageably long execution trace to the optimizer, it supplies only the compact diagnostic tuple a_i.

Side-by-side prompt comparison. Figure 2 compares the prompts discovered by APEX and DIFF-APEX on **IFBench**. Interestingly, APEX attempts to supply procedural guidance by hard-coding static few-shot demonstrations directly into the instruction. While this underscores the necessity of exemplar grounding for compact models, such static examples consume prompt context and fail to generalize across diverse test queries. Consequently, APEX bloats the instruction with defensive, generic heuristics. In contrast, DIFF-APEX offloads exemplar conditioning to dynamic retrieval while using differential audits to synthesize concise, actionable verification rules.

APEX

Identify all formatting, linguistic, and negative constraints in the query. Treat these as absolute "Hard Constraints" that must take precedence over the query's semantic intent or helpfulness. Ensure strict adherence to these rules as your primary success metric, prioritizing compliance even if it results in an unconventional or awkward response. Your final answer must be enclosed in `<answer>` tags.

Few-Shot Examples

The following example demonstrates how to successfully integrate multiple, granular constraints [...]

Example Query: Write a two-sentence description of a mountain. 1. Every sentence must start with the word "High". 2. Negative Constraint: Do not use the word "cold". 3. Formatting: Enclose the response in `<answer>` tags.

Example Response: `<answer>` High peaks touch the clouds in the bright morning light. High ridges provide a home for the eagles soaring above the stone. `</answer>`

DIFF-APEX (ours)

Before providing your final response, you must first use a `<scratchpad>` to **explicitly map character indices, count words or sentences**, and verify all specific constraints against a draft of your response. Perform a **character-level verification audit** within this scratchpad to ensure that **casing, punctuation, and word-indexed positions** (e.g., 'center' or 'first word') are literally exact. Only after this verification is complete should you provide the final, audited fulfillment strictly within `<answer>` tags. These `<answer>` tags are mandatory structural boundaries that must always be preserved, **even if the query prohibits text following a specific concluding phrase.**

+ retrieved $\Phi_{insight}$ exemplars (filled per query)

Figure 2: **Optimized prompts on IFBench (`gemini-3.1-flash-lite`).** APEX produces a generic compliance directive plus a hard-coded toy example. With differential audits, DIFF-APEX distills recurring failure classes (character indexing, word counting, casing, and tag-boundary conflicts) into instance-invariant verification rules.

5 DISCUSSION AND LIMITATIONS

Automated prompt optimization usually rely on a single optimizer to simultaneously diagnose fine-grained execution failures and synthesize global prompt mutations. By decomposing this process into two distinct roles (a teacher that isolates intermediate reasoning divergences and an optimizer that aggregates these diagnostic rules across batches) DIFFPO resolves the bottleneck without saturating the optimizer's context window with noisy failure traces. Importantly, the differential audit is search-agnostic: the diagnostic tuple $\mathbf{a}_i = (d_i, r_i)$ functions as an informational drop-in replacement for binary outcome feedback across reflective optimizers. Furthermore, this design bridges the gap between static rule induction and dynamic in-context learning: indexed procedural blueprints (τ_j) provide adaptive guidance on demand, while synthesized prompt directives actively prevent compact models from copying demonstration artifacts.

Limitations. Several practical constraints remain. First, compiling the auxiliary strategy index $\mathcal{KB}$ incurs front-loaded teacher compute during the offline phase. While this investment is strictly non-recurring and introduces zero teacher queries or extra latency at test time, it assumes access to an auxiliary pool of domain-relevant problems $\mathcal{D}_{\text{pool}}$. Second, the precision of the differential audit relies on the teacher model outperforming the student. Finally, our current study evaluates single-turn reasoning tasks in mathematics, logic, and instruction following; extending differential trace auditing to interactive multi-turn dialogues remains an open direction for future work.

6 CONCLUSION

We introduced DIFFPO, an automated prompt optimization framework that utilizes a stronger model to design prompts for smaller model by pairing teacher-guided differential auditing with structured strategy indexing. Across mathematical reasoning (**MATH-`Hard`**), logical puzzles (**CodeIO-`Puzzle`**), and instruction following (**IFBench**), DIFFPO consistently outperforms existing baselines. Our findings demonstrate that targeted, asymmetric teacher supervision offers an effective path toward closing the reasoning gap for compact models.

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. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*, 2025.

Parth Asawa, Alan Zhu, Abigail O'Neill, Matei Zaharia, Alexandros G Dimakis, and Joseph E Gonzalez. How to train your advisor: Steering black-box llms with advisor models. *arXiv preprint arXiv:2510.02453*, 2025.

Sanket Badhe and Deep Shah. Prompt-level distillation: A non-parametric alternative to model fine-tuning for efficient reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 2131–2147, 2026.

Emmanuel Aboah Boateng, Cassiano O Becker, Nabiha Asghar, Kabir Walia, Ashwin Srinivasan, Ehi Nosakhare, Soundararajan Srinivasan, and Victor Dibia. Concept distillation from strong to weak models via hypotheses-to-theories prompting. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 638–654, 2025.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. Black-box prompt optimization: Aligning large language models without model training. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3201–3219. Association for Computational Linguistics, August 2024.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1107–1128. Association for Computational Linguistics, November 2024.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 21–27 Jul 2024.

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

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

Shanu Kumar, Akhila Yesantarao Venkata, Shubhanshu Khandelwal, Bishal Santra, Parag Agrawal, and Manish Gupta. Sculpt: Systematic tuning of long prompts. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14996–15029, 2025.

Junlong Li, Daya Guo, Dejian Yang, Runxin Xu, Yu Wu, and Junxian He. Codei/o: Condensing reasoning patterns via code input-output prediction. *arXiv preprint arXiv:2502.07316*, 2025.

Guanbao Liang, Yuanchen Bei, Sheng Zhou, Yuheng Qin, Huan Zhou, Bingxin Jia, Bin Li, and Jiajun Bu. Generalizable self-evolving memory for automatic prompt optimization. *arXiv preprint arXiv:2603.21520*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Wenjin Liu, Haoran Luo, Xueyuan Lin, Haoming Liu, Tiesunlong Shen, Jiapu Wang, Rui Mao, and Erik Cambria. Prompt-r1: Collaborative automatic prompting framework via end-to-end reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16260–16280, 2026.

Xinyu Lu, Kaiqi Zhang, Jinglin Yang, Boxi Cao, Yaojie Lu, Hongyu Lin, Min He, Xianpei Han, and Le Sun. Pˆ 2o: Joint policy and prompt optimization. *arXiv preprint arXiv:2603.21877*, 2026.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 11048–11064, 2022.

Zehua Pei, Hui-Ling Zhen, Shixiong Kai, Sinno Jialin Pan, Yunhe Wang, Mingxuan Yuan, and Bei Yu. Scope: Prompt evolution for enhancing agent effectiveness. *arXiv preprint arXiv:2512.15374*, 2025.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. GrIPS: Gradient-free, edit-based instruction search for prompting large language models. In Andreas Vlachos and Isabelle Augenstein (eds.), *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3845–3864. Association for Computational Linguistics, May 2023.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

Prith Sharma and Austin Z Henley. Modular prompt optimization: Optimizing structured prompts with section-local textual gradients. *arXiv preprint arXiv:2601.04055*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Fei Wang, Si Si, Cho-Jui Hsieh, and Inderjit S Dhillon. Apex: Automated prompt engineering expert with dynamic data selection. *arXiv preprint arXiv:2606.11459*, 2026.

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

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

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

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Chenyi Zhou, Zhengyan Shi, Yuan Yao, Lei Liang, Huajun Chen, and Qiang Zhang. Riot: Efficient prompt refinement with residual optimization tree. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22307–22323, 2025.

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

A AI DISCLOSURE

In this work, we used generative AI tools to refine the text, correct grammatical errors and organize the tables and figures throughout the manuscript, and assist in implementing two of the baseline method (PLD and CD). We did not use generative AI to help develop theoretical models or conceptual frameworks, design or critique research methodologies and experiments, conduct qualitative or thematic data analysis, or interpret results; all other disclosure categories specified by the venue are not applicable to this work. We have reviewed all AI-assisted output and take full responsibility for the final content of this paper, including any text, claims, or artifacts produced with the aid of generative AI.

B ALGORITHM

Algorithm 1 DIFFPO Optimization Procedure

Require: Auxiliary pool $\mathcal{D}_{\text{pool}}$, training set $\mathcal{D}$, student model $\mathcal{M}_S$, teacher model $\mathcal{M}_T$, optimizer OPT, dense encoder E, initial instruction π_0, similarity threshold γ, retrieval count K, iterations T.

// Phase 1: Offline Strategy Index Construction (§3.2)
$\mathcal{KB} \leftarrow \emptyset$
for $(x_j, y_j) \in \mathcal{D}_{\text{pool}}$ **do**
  $(s_j, \tau_j) \leftarrow \mathcal{M}_T(x_j, y_j)$ ▷ Synthesize reference solution and abstraction
  if s_j terminates in y_j **then**
    $\mathbf{v}_j \leftarrow E(\tau_j)$ ▷ Precompute structural key embedding
    $\mathcal{KB} \leftarrow \mathcal{KB} \cup \{(x_j, \tau_j, s_j, y_j, \mathbf{v}_j)\}$
  end if
end for

// Phase 2: Offline Reference Caching (§3.5)
for $(x_i, y_i) \in \mathcal{D}$ **do**
  Cache reference solution $s_i \sim \mathcal{M}_T(\cdot \mid x_i)$
end for

// Phase 3: Differential Prompt Optimization
$\Pi \leftarrow \{\pi_0\}$
for $t = 1$ **to** T **do**
  Sample evaluation batch $\mathcal{B} \subset \mathcal{D}$; $\mathcal{A} \leftarrow \emptyset$
  for $\pi \in \Pi$ **do**
    for $(x_i, y_i) \in \mathcal{B}$ **do**
      $\mathcal{N}_K(x_i) \leftarrow \left\{ j \in \text{Top-}K(\text{Sim}(x_i, \cdot)) \;\middle|\; \text{Sim}(x_i, j) \geq \gamma \right\}$ ▷ Threshold gating
      $(\hat{s}_i, \hat{y}_i) \sim \mathcal{M}_S\big(\cdot \mid x_i;\, \pi,\, \Phi_{\text{insight}}(\mathcal{N}_K(x_i))\big)$ ▷ Execute student
      if $\mathcal{U}(\hat{y}_i, y_i) = 0$ **then**
        $a_i \leftarrow \mathcal{M}_{T\text{audit}}(x_i, s_i, \hat{s}_i)$ ▷ Diagnose failure: (d_i, r_i)
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{a_i\}$
      end if
    end for
    $\mathcal{L}(\pi) \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{U}(\hat{y}_i, y_i)$ ▷ Compute batch accuracy
  end for
  $\Pi' \leftarrow \text{OPT}\big(\Pi, \mathcal{A}, \mathcal{L}\big)$ ▷ Propose revisions from frequency-ranked rules
  $\Pi \leftarrow$ top N instructions from $\Pi \cup \Pi'$ ranked by $\mathcal{L}$
end for
return $\arg\max_{\pi \in \Pi} \mathcal{L}(\pi)$

C HYPERPARAMETER DETAILS

C.1 KNOWLEDGE BASE CONSTRUCTION.

For each benchmark, the auxiliary strategy pool $\mathcal{D}_{\text{pool}}$ is constructed to be strictly disjoint from both the prompt optimization batch $\mathcal{B} \subset \mathcal{D}$ and the test split. For **MATH-Hard**, $\mathcal{D}_{\text{pool}}$ is populated from

the remaining problems in the original MATH dataset (Hendrycks et al., 2021) not present in our training or test sets. For **CodeIO-`Puzzle`** and **IFBench**, we randomly sample problems for index initialization. Following the offline pipeline described in §3.2, teacher solutions and hierarchical blueprints are generated using `gemini-3-flash`. In all cases, candidate entries whose generated teacher solutions fail to terminate in the ground-truth answer y_j are discarded.

C.2 Exemplar Representation

Format	Contents	Operational Rationale
Φ_{full}	(x_j, s_j, y_j)	**Standard Exemplar Baseline:** Reference problem, concrete solution, and final answer verbatim.
Φ_{micro}	(τ_j)	**Abstract Procedure Alone:** Omits concrete solutions and values; provides purely algorithmic scaffolding to prevent constant bleed.
Φ_{insight}	(τ_j, x_j, s_j, y_j)	**Hierarchical Guidance:** Articulates domain context before execution steps, maximizing structural grounding.

Table 5: Summary of exemplar representation formats comprising domain context τ_j, problem x_j, solution s_j, and reference answer y_j.

C.3 Optimization Baselines and Hyperparameter Configuration

To ensure an equitable and reproducible comparison across all evaluated methodologies, all algorithms share a standardized teacher–student backbone, identical dataset splits, and unified evaluation protocols across all benchmark tasks. Hyperparameter selection and model checkpoints are governed by deterministic held-out validation partitions; $N_{\text{val}} = 100$ for all datasets.

Here we summarize the key hyperparameter values across all optimization frameworks.

Prompt-Level Distillation (PLD) (Badhe & Shah, 2026).

1. **Supervised Instruction Extraction:** The teacher model inspects labeled training problems to generate explicit step-by-step reasoning trajectories and modular executable directives.
2. **Clustering Logic Synthesis:** Vector embeddings of extracted rules are produced using `gemini-embedding-2`. Semantic clustering is performed via DBSCAN under cosine distance with neighborhood radius $\varepsilon = 0.4$ and a density threshold of $n_{\text{min}} = 6$ samples. Dense clusters are synthesized into consolidated behavioral axioms.
3. **Closed-Loop Conflict Resolution:** Synthesized prompts undergo closed-loop refinement for up to $T_{\text{max}} = 20$ iterations. Each step evaluates $N_{\text{train}} = 50$ training problems, providing $k_{\text{feedback}} = 5$ paired positive and negative demonstrations to the conflict synthesizer.

Concept Distillation (CD) (Boateng et al., 2025).

1. **Error Discovery (Phase 1):** The student $\mathcal{M}_S$ is evaluated with the initial baseline prompt p_0, capturing up to $M_{\text{max}} = 60$ misclassified samples.
2. **Rule Induction (Phase 2):** For each student mistake, the teacher model executes a two-turn dialogue: first diagnosing $n_{\text{reasons}} = 5$ underlying root causes, followed by inducing $n_{\text{concepts}} = 5$ generalized rules.
3. **Deduction and Verification (Phase 3):** The candidate rules are deduplicated and verified on the validation split under a strict non-regression improvement criterion ($\Delta_{\text{acc}} > 0$) or an absolute accuracy barrier $\tau_{\text{verify}} = 0.80$. At most $C_{\text{max}} = 15$ verified concepts are incorporated into the final prompt.

APEX and GEPA

1. **Search and Population Dynamics:** APEX maintains a population pool of $P = 5$ prompt candidates with a beam expansion factor of $B = 5$ (for **MATH-`Hard`** and **IFBench**) and

$B = 10$ for **CodeIO-`Puzzle`**, optimized over $T_{\text{max}} = 20$ iterations for APEX/GEPA and 10 for DIFF-APEX /DIFF-GEPA .