GRAFT: TRANSFORMATION-LEVEL CREDIT ASSIGNMENT OVER PROMPT DERIVATION GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite rapid advances in large language model (LLM) capabilities, effective prompting remains essential for eliciting their full potential, particularly in compound systems governed by multiple interacting instructions. Automatic prompt optimization (APO) automates this process, yet existing methods organize search around prompt candidates and underuse evidence about the transformations that produce them, leading to inefficient exploration under limited budgets. We introduce **G**raph-based C**R**edit **A**ssignment **F**or Prompt **T**ransformations (GRAFT), built on the insight that a proposal's evaluation provides evidence not only about the resulting prompt but also about the realized transformation, making the transformation the natural unit of search and credit assignment. GRAFT records optimization as a multi-parent prompt derivation graph and maintains Monte Carlo statistics on transformation edges, allowing accumulated evidence to guide subsequent search toward productive transformations. To capture different notions of transformation utility, we design two reward formulations for GRAFT: mean reward measures average performance, while coverage reward measures how frequently a transformation's output reaches the current per-instance frontier. Across six benchmarks and two open-source LLMs, GRAFT achieves the strongest aggregate performance; on Qwen3-8B, it outperforms GEPA, the strongest baseline, by 3.14 points while using 4.50% fewer rollouts and 20.0% fewer tokens.

1 INTRODUCTION

Even as the capabilities of large language models (LLMs) continue to advance, effective prompting remains crucial for eliciting strong performance on specific tasks, particularly in compound LLM systems where the instructions governing individual modules can substantially affect overall behavior. Compared with fine-tuning, automatic prompt optimization (APO) provides a lightweight and flexible way to adapt LLM-based systems to specific tasks, often requiring less compute and enabling faster iteration without modifying model parameters. APO formulates prompt design as black-box optimization over natural-language instructions: an optimizer proposes candidate prompts, evaluates the resulting system on task examples, and uses the observed scores to guide subsequent proposals (Zhou et al., 2023; Yang et al., 2024; Pryzant et al., 2023). This propose-evaluate-update loop provides a general interface for optimizing both individual prompts and the instructions of multi-module LLM systems (Khattab et al., 2024; Opsahl-Ong et al., 2024).

To improve the effectiveness of APO under limited evaluation budgets, recent work has focused on combining richer feedback with more structured exploration. Reflection-based methods convert model errors or execution traces into targeted prompt revisions, allowing each rollout to provide natural-language guidance beyond a scalar score (Pryzant et al., 2023; Agrawal et al., 2026). Evolutionary and tree-search methods complement this feedback by maintaining multiple candidates, revisiting promising search trajectories, and generating new candidates through mutation and crossover, thereby exposing multiple transformation paths within a single optimization run (Guo et al., 2023; Wang et al., 2024).

Despite these advances, the search state of current APO methods is still centered on prompt candidates. Evaluation outcomes are used to rank candidate prompts and, in some methods, to update operator-level preferences, but they do not quantify the value of a transformation together with the context in which it was applied and the complete set of candidates from which it was derived. As

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.

optimization proceeds, evidence about which edits work in different regions of the prompt space and which candidate pairs recombine effectively does not become reusable search statistics. Consequently, key decisions such as operator selection, crossover timing, and partner selection cannot directly exploit the transformation history accumulated during search. This can waste the limited rollout budget on unproductive edits and recombinations.

In this paper, we propose **G**raph-based C**R**edit **A**ssignment **F**or Prompt **T**ransformations (GRAFT), a reflective prompt optimizer built on a simple premise: a proposal's score provides evidence not only about the resulting prompt, but also about the transformation that produced it. GRAFT represents optimization history as a prompt derivation graph, where prompt artifacts are vertices and mutation or crossover transformations are explicit edges that may connect multiple parents to a child. By maintaining Monte Carlo statistics on these edges, GRAFT allows each transformation to accumulate evidence from subsequent exploration and turns this evidence into actionable search decisions. Specifically, GRAFT learns which operator to apply, when to invoke crossover, and which complementary candidate to use as the crossover partner. To define what constitutes a valuable transformation, we design two reward formulations. Mean credit favors edits that improve average performance across examples, such as a broadly useful output-format rule, whereas coverage credit rewards per-instance frontier coverage and preserves edits that solve complementary subsets, such as distinct multi-hop reasoning patterns or mathematical problem types.

Our contributions can be summarized as follows:

1. We introduce GRAFT, a graph-based APO framework that treats each realized prompt transformation as the basic unit of search and credit assignment. Its multi-parent derivation graph preserves the relations among source prompts, optimization actions, and resulting artifacts, allowing transformation-level credit to guide which operator to apply, when to invoke crossover, and which candidate to select as the crossover partner.
2. We further design mean and coverage rewards for transformation-level search: mean reward favors transformations whose improvements are broadly shared across task instances, while coverage reward preserves transformations that contribute gains on complementary instance subsets. Together, they support search across tasks with different improvement structures.
3. Experiments on six benchmarks and two mainstream LLMs show that GRAFT outperforms strong reflective, evolutionary, and tree-search APO methods at matched or lower rollout budgets. Ablations further demonstrate that GRAFT's core design choices, including transformation-level credit assignment and its reward formulations, are critical to these gains.

2 Related Works

Reflective and evolutionary prompt optimization. Early APO methods propose, score, and select instructions: APE (Zhou et al., 2023) samples candidates from an LLM, OPRO (Yang et al., 2024) continues a solution-score history placed in a meta-prompt, and ProTeGi (Pryzant et al., 2023) edits along natural-language "gradients" derived from minibatch failures under beam search. For compound systems, DSPy (Khattab et al., 2024) and MIPROv2 (Opsahl-Ong et al., 2024) optimize instructions and demonstrations under a downstream metric, while TextGrad (Yuksekgonul et al., 2025) and Trace (Cheng et al., 2024) propagate textual feedback through a computation graph. Evolutionary methods instead maintain a population: EvoPrompt (Guo et al., 2024) simulates genetic and differential-evolution operators through an LLM, PromptBreeder (Fernando et al., 2024) additionally evolves its own mutation prompts, and GEPA (Agrawal et al., 2026) reflects on execution traces and samples the next candidate from a per-instance coverage frontier. Across this line of work, optimization history is organized primarily around candidate quality, program parameters, or proposal policies. GRAFT instead makes each realized transformation a persistent object of search and credit assignment, allowing successful edits to guide subsequent exploration.

Structured search and adaptive proposal selection. PromptAgent (Wang et al., 2024) casts prompt optimization as an MDP whose states are prompts and whose actions are error feedbacks, and searches it with MCTS. MCTS-AHD (Zheng et al., 2025) organizes LLM-generated heuristics

in a tree and uses progressive widening with mutation and crossover actions. Its crossover draws material from multiple subtrees but attaches the resulting child to a virtual root, leaving the contributing set outside the searched relation. In parallel, OPTS (Ashizawa et al., 2025) and RIDER (Dragomirov et al., 2026) adapt prompt strategies or operators through Thompson sampling, while HAPO (Chen et al., 2026) maintains UCB statistics for semantic-unit and edit-operation pairs. GRAFT brings structured search and adaptive selection together in a prompt derivation graph, where each realized, possibly multi-parent transformation accumulates statistics that guide operator choice, crossover timing, and partner selection, concentrating rollouts on promising transformations.

Lineage- and graph-structured optimization. Recent methods increasingly retain explicit lineage or graph structure during optimization. SEPO (Ma et al., 2026) applies local edits to a typed prompt schema and records the examples each edit fixes or breaks as textual lineage feedback. PathWise (Gungordu et al., 2026) constructs multi-parent entailment hyperedges that preserve derivation rationales and performance history, and navigates the resulting graph with policy, world-model, and critic agents. DGA2D (Zhao et al., 2026) assigns context-dependent credit to transitions between code implementations within algorithm execution graphs. GRAFT couples complete multi-parent prompt derivations with persistent Monte Carlo transformation credit, using the same graph to determine where to search, which operator to apply, and which candidates to combine, turning lineage into actionable search guidance.

3 METHOD

GRAFT optimizes prompts by explicitly representing the transformations that connect candidate artifacts and assigning credit to those transformations. The overall architecture of GRAFT is illustrated in Figure 1. Each iteration traverses the resulting derivation graph, applies mutation or crossover to create a new candidate, evaluates the candidate, and uses its reward to update the search statistics. We first formulate the APO problem, then present the graph representation, the search procedure, and the transformation-level reward and backup rule.

3.1 PROBLEM FORMULATION

We consider a compound AI system Φ composed of $|M|$ language modules $M = \langle M_1, \ldots, M_{|M|} \rangle$ orchestrated by control flow C, where each module M_i carries an instruction π_i and frozen weights θ_i. Writing $\Pi_\Phi = \langle \pi_1, \ldots, \pi_{|M|} \rangle$ for the collection of module instructions, we optimize Π_Φ only and hold Θ_Φ fixed. A task instance is a pair (x, m) in which x conforms to the system's input schema and m holds evaluator metadata such as a gold answer, a rubric, or unit tests. A metric $\mu : \mathcal{Y} \times \mathcal{M} \to [0, 1]$ scores a system output against that metadata. Following GEPA, we also assume a feedback function μ_f that returns, alongside the scalar score, the textual execution and evaluation traces produced during a rollout.

The optimizer is given a seed system, a train split $\mathcal{D}_{\text{train}}$ partitioned into a reflection split $\mathcal{D}_{\text{fb}}$ and a selection split $\mathcal{D}_{\text{val}}$, and a rollout budget B, where one rollout is one invocation of Φ together with its evaluation by μ:

$$\Pi_\Phi^\star = \arg\max_{\Pi_\Phi} \; \mathbb{E}_{(x,m)\sim\mathcal{T}} \left[\mu \left(\Phi \left(x; \langle \Pi, \Theta_{\text{frozen}} \rangle_\Phi \right), m \right) \right], \qquad \text{s.t.} \quad \#\text{rollouts} \le B.$$

Beyond the aggregate score of a candidate, we retain its **per-instance score vector** over the selection split, $S(\Pi) = [s_1(\Pi), \ldots, s_n(\Pi)]$, with $s_i(\Pi) = \mu(\Phi(x_i; \Pi), m_i)$ and $n = |\mathcal{D}_{\text{val}}|$. Retaining the vector rather than its mean is necessary because candidates in this search space are routinely complementary: a candidate with an unremarkable average may be the best available on a hard or long-tailed subset of instances, and an aggregate score cannot express that.

The difficulty of this problem is not only that the search space is discrete and generated by an LLM, but that each rollout must inform three coupled decisions: which existing candidate to extend, which operator to apply, and, if that operator recombines, which partner to select. Candidate scores directly reveal which artifacts perform well, but only indirectly characterize the transformations that produced them. We therefore study how an optimizer can convert artifact-level evaluations into persistent evidence about its transformations and use that evidence to guide subsequent search.

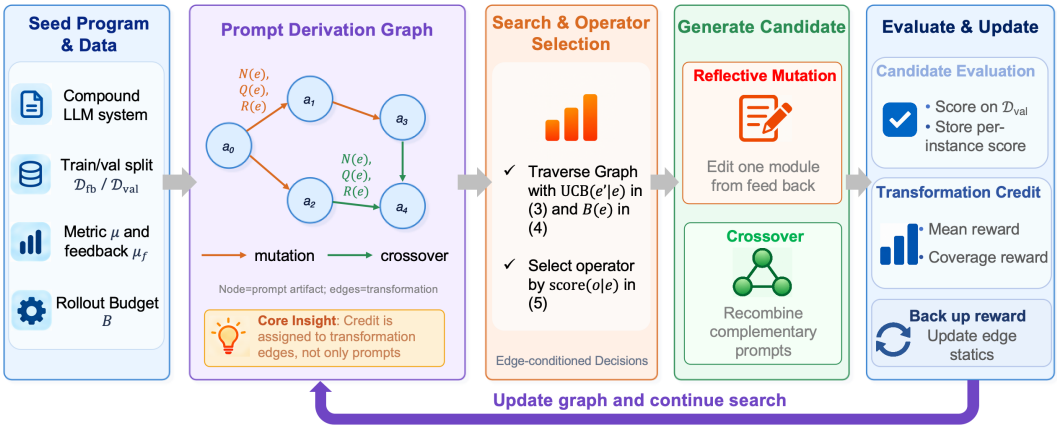


Figure 1: Overall architecture of GRAFT. Edge-conditioned statistics guide graph traversal and the choice between reflective mutation and crossover. Candidate rewards are assigned to the responsible transformations and backed up to guide subsequent search.

3.2 Prompt Derivation Graph

GRAFT maintains optimization history as a directed acyclic derivation graph $G = (V, E)$. Each vertex stores a prompt artifact, and each edge records one realized transformation together with its parent artifacts, operator, and resulting child. Transformations with multiple parents are represented as directed hyperedges; for readability, we refer to both ordinary and multi-parent relations as edges.

A **prompt artifact** is a unique candidate system, recorded as

$$a = \big(\text{id},\ \Pi,\ h(\Pi),\ S(\Pi),\ H(\Pi)\big), \tag{1}$$

where $h(\Pi)$ is a hash used for exact deduplication, $S(\Pi)$ is the per-instance score vector on $\mathcal{D}_{\text{val}}$, and $H(\Pi)$ is the module history recording which modules differ from the seed system. A **transformation edge** records one derivation,

$$e = \big(\mathcal{A}_{\text{parent}},\ o,\ a_{\text{child}},\ e_{\text{source}},\ e_{\text{partner}}\big), \tag{2}$$

with $\mathcal{A}_{\text{parent}}$ the set of parent artifacts, o the operator, a_{child} the produced artifact, and $e_{\text{source}}, e_{\text{partner}}$ the edges from which the source and partner artifacts were selected. An **init edge** has no parent and points to the seed system. A **mutation edge** $e = (\{a_p\}, \text{mutation}, a_c)$ rewrites one module of a single parent, while a **crossover edge** $e = (\{a_1, a_2\}, \text{crossover}, a_c)$ recombines module instructions from two parents. Each edge carries its own search statistics, $N(e)$, $Q(e)$ and $R_{\text{sum}}(e)$, where $N(e)$ is the visit count and $R_{\text{sum}}(e)$ is the cumulative backed-up reward. Writing the parent–operator–child component of a realized incoming transformation as $e_t \colon \mathcal{A}_{t-1} \xrightarrow{o_{t-1}} a_t$, we interpret $Q(e_t) = R_{\text{sum}}(e_t)/N(e_t) \approx \mathbb{E}[G_t \mid e_t]$, where G_t denotes the backed-up downstream return. Thus, $Q(e_t)$ evaluates the specific transformation context rather than the child artifact alone.

Artifacts store candidate programs and their evaluations, whereas visit and value statistics are attached to derivation edges. The artifact and edge definitions in equation 1 and equation 2 preserve two forms of derivation information. A crossover edge records both parents as part of the same transformation, and exact deduplication by $h(\Pi)$ allows distinct transformations to terminate at the same artifact while retaining separate search statistics. GRAFT can therefore estimate the downstream value of each transformation context even when the resulting artifact is shared.

3.3 Search over the Derivation Graph

Each search iteration selects a transformation edge to expand, chooses mutation or crossover, and generates a new candidate for evaluation. GRAFT balances promising transformations against under-explored alternatives while controlling the branching induced by LLM-generated proposals.

Graph traversal. Starting from the root edge, GRAFT descends to successor edges according to the edge-level UCB rule in equation 3:

$$\mathrm{UCB}(e' \mid e) = Q(e') + c\sqrt{\frac{\log\big(N(e)+1\big)}{N(e')+1}}, \tag{3}$$

where e is the current edge and e' a successor. Descent stops, and the current edge is expanded, as soon as that edge still has expansion budget.

Progressive widening (PW). Because an LLM can propose unboundedly many distinct successors, we cap the number a given edge may accept using the visit-dependent quota in equation 4:

$$B(e) = \max\left\{1, \lfloor k\,(N(e)+1)^{\alpha} \rfloor\right\}, \qquad \text{expand } e \text{ only if } |\mathrm{Succ}(e)| < B(e). \tag{4}$$

The minimum quota of one ensures that every newly created edge can be expanded at least once. We additionally impose a failure cap proportional to the widening quota, which prevents an edge from consuming budget on repeated invalid or rejected proposals. Together, UCB selection and progressive widening allocate more expansion opportunities to transformations with strong or uncertain downstream value.

Operator selection. For each edge e and operator o, we maintain an edge-conditioned visit count and advantage estimate, $N_{\mathrm{op}}(e,o)$ and $Q_{\Delta}(e,o)$, where the advantage is measured as improvement over the parent rather than as absolute score. The operator applied when expanding e maximizes the edge-conditioned score in equation 5:

$$\mathrm{Score}(o \mid e) = Q_{\Delta}(e,o) + c_{\mathrm{op}}\sqrt{\frac{\log\big(N(e)+1\big)}{N_{\mathrm{op}}(e,o)+1}} + \eta \cdot \mathbf{1}\big[\mathrm{plateau}(e)\big] \cdot \mathbf{1}\big[o = \mathrm{crossover}\big], \tag{5}$$

where $\mathrm{plateau}(e)$ fires when the recent downstream improvements along e have stayed below a threshold. Whereas $Q(e)$ evaluates the realized transformation leading to the current artifact, $Q_{\Delta}(e,o)$ estimates the immediate advantage of applying the next operator o from that transformation context. The third term encodes the intuition that once local editing along a lineage has stopped paying, the productive move is to import a complementary strategy from another branch rather than to keep mutating.

Reflective mutation. Mutation reuses GEPA's reflective update, applied in the context of a source edge e with child artifact a_p. First, a module is selected by a per-edge round-robin pointer, so that module update order is local to the selected edge. A minibatch of size b is then drawn from $\mathcal{D}_{\mathrm{fb}}$, and the parent is executed to collect module-level traces and metric feedback via μ_f. A reflection LM receives the current instruction, traces, feedback, and scores, and proposes a replacement. The resulting child is screened on an independently sampled gate minibatch of size g from $\mathcal{D}_{\mathrm{fb}}$. Mutation compares the child with its parent on the same gate examples, whereas crossover compares it with the better of its two parents. Only proposals with positive gate improvement proceed to full evaluation on $\mathcal{D}_{\mathrm{val}}$, and all gate task-model evaluations count toward the optimization budget. Evaluated candidates are admitted to the graph when $\Delta_{\mathrm{val}} \geq -\epsilon_{\mathrm{tree}}$, retaining near-neutral stepping stones for subsequent mutation or recombination.

Crossover partner selection. When crossover is selected, GRAFT scores candidate partners from the current coverage edge frontier, defined in Section 3.4, and the top-K accepted edges by child score. We exclude the source edge, edges without a score vector, pairs that have reached the crossover cap, and artifacts that are too similar to the source. For a source edge e_s and candidate partner e_p, we define **complementarity** as the mass by which the partner exceeds the source per instance, $\mathrm{Comp}(e_s,e_p) = \frac{1}{n}\sum_{i=1}^{n}\max\big(0,\ S_{e_p}[i] - S_{e_s}[i]\big)$, and select the partner according to equation 6:

$$\begin{aligned}\mathrm{PartnerScore}(e_s,e_p) = {} & \lambda_q Q(e_p) + \lambda_s \mathrm{Score}(a_{e_p}) \\ & + \lambda_c \mathrm{Comp}(e_s,e_p) + \lambda_u \frac{1}{N(e_p)+1} + \lambda_p Q_{\mathrm{partner}}(e_p).\end{aligned} \tag{6}$$

Here, $\mathrm{Score}(a_{e_p})$ is the aggregate selection score of the partner artifact, and the λ terms are weighting coefficients. The five terms in equation 6 prefer partners that are respectively promising downstream, individually strong, complementary on the source's weak instances, under-explored, and historically useful as partners.

Module-aware crossover. Crossover recombines the source edge's child with the selected partner's child at module granularity. The child is initialized from the source program; then, for each module, it inherits the partner's instruction if the partner evolved that module and the source did not, keeps the source's instruction in the symmetric case, chooses uniformly at random between them if both evolved the module to different instructions, and retains the seed instruction if neither did. This module-wise rule preserves useful instructions while selectively importing complementary improvements from another lineage. The resulting program follows the same gate and selection-split evaluation as mutation, with the gate and selection baselines set to the maximum score of the two parents.

3.4 Transformation Credit and Backup

Evaluating the child of a transformation edge e produces a per-instance score vector S_e. We convert this vector into a scalar reward using two formulations that capture different notions of transformation utility.

Mean reward. The mean formulation measures the aggregate quality of the child artifact,

$$R_{\text{mean}}(e) = \frac{1}{n}\sum_{i=1}^{n} S_e[i]. \tag{7}$$

It favors transformations whose children perform well across the selection instances.

Coverage reward. The coverage formulation measures how much of the current per-instance frontier is covered by the child. Let $b_i = \max_{e' \in \mathcal{E}_{\text{acc}}} S_{e'}[i]$ be the best score attained by any accepted edge on instance i. The coverage edge frontier is $\mathcal{F} = \{e \in \mathcal{E}_{\text{acc}} : \exists i,\ S_e[i] \geq b_i - \epsilon\}$, and the reward is the fraction of instances on which e reaches the frontier,

$$R_{\text{coverage}}(e) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\big[S_e[i] \geq b_i - \epsilon\big]. \tag{8}$$

Together, equation 7 and equation 8 assess transformation utility through average score magnitude or frontier attainment. Because the frontier changes as search progresses, GRAFT recomputes b_i and refreshes the affected rewards and running estimates whenever the frontier advances.

Backup. After a new edge e_{new} has been created and evaluated, GRAFT updates the root edge, the edges on the selected descent path, the source edge, and the new edge itself. For each such edge,

$$N(e) \leftarrow N(e) + 1, \qquad Q(e) \leftarrow Q(e) + \frac{R - Q(e)}{N(e)}. \tag{9}$$

In equation 9, R is the reward of the new edge under the chosen formulation. The reward is evaluated on the resulting child, but credit is assigned to the transformation edges traversed during search rather than to artifact nodes. A crossover is credited as a single multi-parent transformation, while the selected partner's historical utility is tracked separately through Q_{partner}. When a proposal never reaches a selection-split evaluation, for example because it failed the gate, we increment the visit counts of the affected search edges but leave their value estimates unchanged. The same backup pass refreshes the decision statistics and the recent-improvement window used for plateau detection. This closes the loop from observed transformation outcomes to subsequent search decisions. The complete optimization procedure is summarized in Algorithm 1 in Appendix A.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate GRAFT on six benchmarks spanning three broad capability groups. HotpotQA (Yang et al., 2018) and GPQA Diamond (Rein et al., 2024) test multi-hop and scientific reasoning. IFBench (Pyatkin et al., 2025) and PUPA (Siyan et al., 2025) evaluate instruction following and privacy-aware delegation in compound LLM systems. AIME-2025 (Balunovic et al., 2026) and LiveBench-Math (White et al., 2025) cover competition-level mathematical reasoning. HotpotQA uses exact match, IFBench uses its constraint-satisfaction score, PUPA averages response

Table 1: Benchmark performance across Qwen3-8B and Gemma3-12B-IT. All results are averaged over three random seeds; for the most competitive methods, including GEPA, GRAFT and their variants, we additionally report standard deviations. Green and underlined values denote the best and second-best results. All tasks use no-think decoding except Qwen3-8B on AIME-2025, which uses thinking mode.

Method	HotpotQA	IFBench	PUPA	GPQA	AIME-25	LB-Math	Agg.
Qwen3-8B							
Baseline	37.50	34.18	80.07	36.70	41.80	34.55	44.13
MIPROv2	58.84	33.67	87.33	42.60	51.33	38.39	52.03
PromptAgent	45.83	35.88	88.07	35.00	57.00	38.12	49.98
GEPA	62.83±1.95	35.24±1.44	95.39±1.55	37.50±3.13	53.78±2.77	36.33±3.86	53.51
GEPA+Merge	64.83±0.71	33.33±1.48	93.53±1.57	37.50±3.13	53.78±2.77	36.33±3.86	53.22
GRAFT (avg)	64.67±1.31	**36.62**±1.67	**95.50**±1.09	41.18±2.08	55.34±1.89	36.94±1.75	55.04
GRAFT (coverage)	**66.89**±0.70	35.99±0.43	93.24±1.41	**44.60**±3.12	**58.00**±2.83	**41.22**±1.63	**56.65**
Gemma3-12B-IT							
Baseline	58.84	39.63	70.56	29.41	18.66	36.23	42.22
MIPROv2	57.16	36.23	90.80	33.09	20.00	**41.34**	46.44
PromptAgent	57.84	39.63	79.97	30.88	19.33	36.98	44.11
GEPA	**62.34**±1.15	42.65±0.64	94.48±1.28	32.35±3.24	18.00±2.44	41.32±3.59	48.52
GEPA+Merge	54.66±4.81	42.27±2.19	92.38±1.12	32.35±3.24	18.00±2.44	41.32±3.59	46.83
GRAFT (avg)	56.84±3.73	43.37±0.72	94.45±0.68	35.29±2.12	17.67±0.47	39.47±2.09	47.85
GRAFT (coverage)	60.44±1.19	**44.56**±1.21	**95.20**±0.93	**38.24**±2.09	**20.67**±0.94	40.18±2.81	**49.88**

quality and privacy preservation, and the remaining tasks use their official answer-based metrics. All scores are reported on a 0 to 100 scale, and Aggregate is the unweighted mean over the six test scores.

Baselines. We compare against the unoptimized seed program and three families of strong APO methods: MIPROv2 (Opsahl-Ong et al., 2024), which jointly searches instructions and demonstrations; PromptAgent (Wang et al., 2024), which combines reflective revision with MCTS; and GEPA (Agrawal et al., 2026), which evolves prompts through trace-based reflection and per-instance Pareto selection. We additionally include GEPA+Merge, its crossover-enabled variant. All methods use the same task model, seed program, data splits, and evaluation metric. GPQA Diamond, AIME-2025, and LiveBench-Math use single-module programs; because GEPA's merge operator recombines instructions across modules, GEPA+Merge reduces to GEPA on these tasks.

Implementation Details. We conduct two sets of experiments across all six benchmarks using Qwen3-8B (Yang et al., 2025) and Gemma3-12B-IT (Team et al., 2025) as task models, covering two widely used open-source model families. Gemini 3.1 Pro serves as the reflection model throughout. All evaluations use no-think decoding except Qwen3-8B on AIME-2025, for which thinking mode is enabled. We use one GRAFT configuration across all benchmarks and models. For GRAFT, results are means with standard deviations across repeated runs. Detailed data splits, inference settings, optimization budgets, hyperparameters, baseline configurations, and reproducibility information are provided in Appendix.

4.2 Experimental Results

Table 1 compares GRAFT with the seed program and strong reflective, evolutionary, and tree-search APO baselines. Table 2 compares the optimization budget and token usage of GRAFT with those of GEPA, the most competitive baseline. We report GRAFT under both reward formulations, so that the effect of the transformation reward is separated from the shared graph-search procedure.

GRAFT consistently improves APO across tasks and models. GRAFT achieves the strongest aggregate performance with both task models. GRAFT-coverage reaches 56.65 on Qwen3-8B, improving over the strongest baseline, GEPA, by 3.14 points, and reaches 49.88 on Gemma3-12B-IT, improving over it by 1.36 points. Considering the two reward variants together, GRAFT obtains

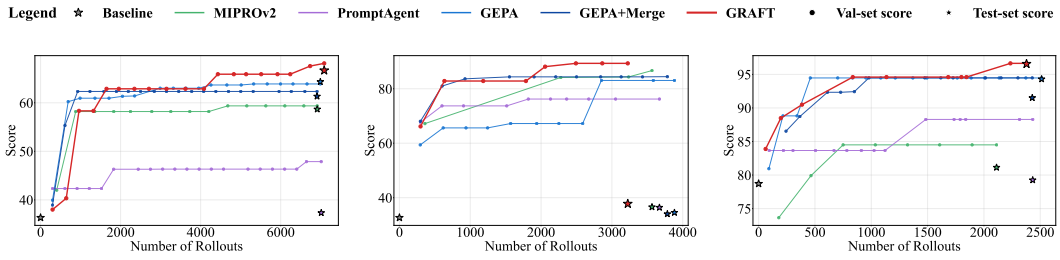
Table 2: Three-seed average optimization cost of GEPA and GRAFT using coverage reward. Rollouts are the total budget, tokens are in millions, and Avg. averages the six benchmarks.

(a) Total optimization budget (# rollouts)

	Qwen3-8B		Gemma-12B	
Benchmark	**GEPA**	**GRAFT**	**GEPA**	**GRAFT**
HotpotQA	7170	6946	6874	6834
IFBench	3805	3600	3776	3724
PUPA	2567	2515	2796	2453
GPQA	1878	1809	1838	1820
AIME-25	1627	1349	1554	1503
LiveBench-Math	1854	1831	1924	1807
Avg.	3150	**3008**	3127	**3023**

(b) Token usage (M)

	Qwen3-8B		Gemma-12B	
Benchmark	**GEPA**	**GRAFT**	**GEPA**	**GRAFT**
HotpotQA	15.8	14.8	17.8	17.3
IFBench	7.9	7.3	7.4	6.9
PUPA	8.7	5.5	19.2	15.2
GPQA	3.6	2.2	3.9	2.5
AIME-25	24.5	18.2	3.4	2.4
LiveBench-Math	4.1	3.7	3.8	3.1
Avg.	10.8	**8.6**	9.3	**7.9**



(a) HotpotQA, Qwen3-8B (b) IFBench, Qwen3-8B (c) PUPA, Qwen3-8B

Figure 2: Optimization curves on three benchmarks with Qwen3-8B; curves show the validation score over cumulative rollouts, and stars mark final test performance. GRAFT achieves the highest validation and test scores under matched or smaller rollout budgets.

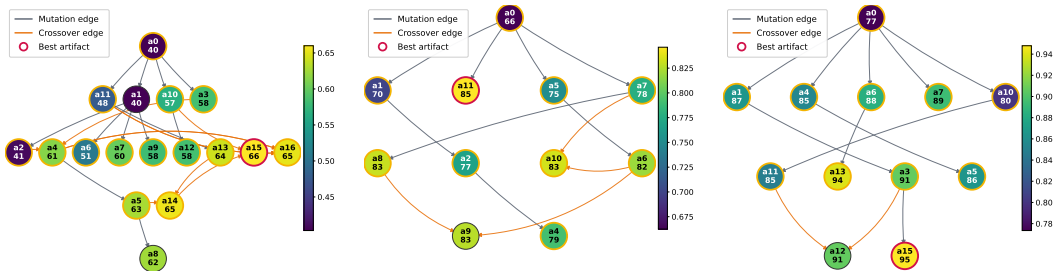
the best result on all six tasks with Qwen3-8B and on four of six tasks with Gemma-12B. The gains span multi-hop and scientific reasoning, instruction following, privacy-aware delegation, and mathematics, indicating that our GRAFT is effective beyond a single task family.

Mean vs. Coverage rewards. Holding the graph search procedure fixed, coverage reward outperforms mean reward on 10 of the 12 model-benchmark combinations and improves aggregate performance by 1.61 points on Qwen3-8B and 2.03 points on Gemma-12B. Its gains are concentrated on HotpotQA, GPQA, AIME, and LiveBench-Math, where it improves over mean reward by 2.86 points on average across the two task models. These benchmarks span diverse knowledge domains, reasoning patterns, and mathematical problem types, while their evaluation signals are largely answer-level and often sparse. In such settings, useful transformations need not improve the same subset of instances. This pattern is consistent with the intended distinction between the two formulations: mean reward summarizes absolute performance across all selection instances and preserves graded score differences, whereas coverage reward measures whether a candidate reaches the current instance-wise frontier and therefore emphasizes frontier coverage. By contrast, mean reward remains stronger on Qwen3-8B for IFBench and PUPA, where fine-grained evaluation signals and broadly shared formatting or privacy requirements provide a more informative aggregate objective.

Optimization efficiency. Table 2 compares the optimization cost of GRAFT with GEPA under matched rollout caps. GRAFT uses 4.50% and 3.31% fewer rollouts on average with Qwen3-8B and Gemma3-12B-IT, respectively, while also reducing average token usage by 20.0% and 14.6%. These savings are consistent with credit-guided traversal and early gating, which focus expansion on promising transformations and avoid full evaluations of unproductive proposals.

4.3 Optimization Dynamics

Performance over rollouts. Figure 2 traces the validation score discovered as the optimization budget increases. GRAFT continues to find stronger prompts after several baseline trajectories have plateaued, ultimately attaining the highest test score on all three tasks with comparable or fewer



(a) HotpotQA, Qwen3-8B (b) IFBench, Qwen3-8B (c) PUPA, Qwen3-8B

Figure 3: Prompt derivation graphs produced by GRAFT on three benchmarks. Mutation drives broad exploration, while selective crossover connects promising lineages and contributes additional competitive artifacts.

Table 3: Ablations on Qwen3-8B using coverage reward. Each column replaces or removes only the named component from full GRAFT; results are averaged over three seeds.

Benchmark	GRAFT	Artifact-level credit	Fixed-rate crossover	Random partner	No crossover
HotpotQA	**66.89**	65.33	64.83	65.34	62.17
IFBench	**35.99**	34.88	35.31	35.03	34.86

rollouts. IFBench presents a challenging out-of-distribution evaluation: although all methods exhibit a substantial validation-to-test gap, GRAFT retains the strongest test performance, indicating greater robustness of the prompts selected during search.

Search structure. Figure 3 visualizes the prompt derivation graphs produced during optimization. Across tasks, mutation creates broad branching and drives most local improvements, while crossover selectively reconnects lineages after alternatives have emerged. Crossover contributes to the highest-scoring region of the HotpotQA graph, whereas the best artifacts on IFBench and PUPA arise primarily through mutation. Rather than enforcing a fixed operator mixture, GRAFT therefore forms task-dependent search structures in which crossover complements local mutation.

4.4 Ablation Studies

Table 3 shows that full GRAFT performs best on both tasks. Artifact-level credit reduces performance by 1.56 points on HotpotQA and 1.11 points on IFBench, supporting attribution to the transformations that produce candidates rather than to candidates alone. Fixed-rate crossover and random partner selection also yield consistent drops, while removing crossover causes the largest degradation on HotpotQA (4.72 points) and a 1.13-point drop on IFBench. Together, these results indicate that crossover is most effective when both its timing and partner are adapted to the current search context. Detailed ablation configurations are provided in Appendix A.3.

5 Conclusion

We introduced GRAFT, a graph-based APO framework that treats realized prompt transformations as the unit of search and credit assignment. By recording multi-parent derivations and maintaining Monte Carlo statistics on transformation edges, GRAFT turns observed transformation outcomes into reusable guidance for subsequent search. Its mean and coverage rewards capture aggregate quality and per-instance frontier attainment, respectively. Across six benchmarks and two LLMs, GRAFT achieves the strongest performance among strong APO baselines while reducing actual rollout and token usage relative to GEPA. These results support transformation-level credit as a promising foundation for budget-efficient prompt optimization and motivate its extension to richer transformation operators and larger compound LLM systems.

AI USE STATEMENT

In this work, generative AI tools were used to assist with manuscript editing, literature search, figure and table preparation, code development, and experimental feedback. All AI-assisted content was reviewed and verified by the authors. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide the core implementation of GRAFT in the supplementary materials. The experimental setup, including dataset splits, evaluation metrics, model configurations, optimization budgets, and search hyperparameters, is detailed in Appendix A. All experiments are conducted using publicly available benchmarks and task models, with the same seed programs, data splits, and evaluation protocols across the compared methods. We report results averaged over three random seeds and provide detailed baseline configurations to support fair comparison and reproducibility of the reported results.

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 *The Fourteenth International Conference on Learning Representations*, 2026.

Rin Ashizawa, Yoichi Hirose, Nozomu Yoshinari, Kento Uchida, and Shinichi Shirakawa. Bandit-based prompt design strategy selection improves prompt optimizers. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20799–20817, 2025.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Advances in Neural Information Processing Systems*, 38, 2026.

Dongyu Chen, Jian Ma, Xianpeng Zhang, Lei Zhang, Haonan Lu, Chen Chen, Chuangchuang Wang, and Kai Tang. Learning from prompt itself: the hierarchical attribution prompt optimization. *arXiv preprint arXiv:2601.02683*, 2026.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024.

Daglar Dragomirov, Nikita Kulin, Sergey Muravyov, Ilya Makarov, Daniil Sukhorukov, and Mikhail Mozikov. Rider: Evolutionary prompt optimization with adaptive operator selection for software engineering. In *Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering*, pp. 1416–1417, 2026.

Oguzhan Gungordu, Siheng Xiong, and Faramarz Fekri. Pathwise: Planning through world model for automated heuristic design via self-evolving llms. *arXiv preprint arXiv:2601.20539*, 2026.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

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 *The Twelfth International Conference on Learning Representations*, 2024.

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

Xiaoyu Ma, Haoyue Liu, Yiwen Li, Jionghao Zhu, Zhichao Wang, Ye Chen, and Xiaoying Tang. Sepo: Evidence-grounded prompt optimization via structural editing. *arXiv preprint arXiv:2608.28067*, 2026.

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

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7957–7968, 2023.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Li Siyan, Vethavikashini Chithrra Raghuram, Omar Khattab, Julia Hirschberg, and Zhou Yu. Papillon: Privacy preservation from internet-based and local language model ensembles. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3371–3390, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

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.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. Livebench: A challenging, contamination-free LLM benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Jiale Zhao, Zimu Chen, Sirui Mao, Wentao Yang, Yuxiang Bai, and Liyuanjun Lai. Dga _2 d: Directed graph-guided automated algorithm design with large language models. *arXiv preprint arXiv:2608.00700*, 2026.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*, 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 Experimental Details

A.1 Datasets and Evaluation

We follow the benchmark implementations and seed programs released in the official GEPA repositories: `https://github.com/gepa-ai/gepa` and `https://github.com/gepa-ai/gepa-artifact`. Each benchmark is partitioned into a reflection split, a selection split, and a held-out test split; the resulting sizes are reported in Table 4. HotpotQA is divided into 20%, 40%, and 40% reflection, selection, and test partitions before selecting the instances used in our experiments. IFBench uses disjoint portions of its training collection for reflection and selection and retains its official test set. PUPA uses its predefined 111/111/221 split. GPQA Diamond is randomly shuffled and divided into 33%, 33%, and 34% partitions. LiveBench-Math is similarly shuffled and divided into three approximately equal partitions. For AIME, we randomly split the historical validation collection equally between reflection and selection, and evaluate on the 30 AIME-2025 problems with five independent generations per problem.

Across repeated runs, we reshuffle the pooled reflection and selection examples and repartition them while preserving the original split sizes; the test split remains fixed. Reported variation across runs therefore captures both optimizer stochasticity and variation in the reflection and selection partitions.

HotpotQA is evaluated using answer exact match. IFBench uses its constraint-satisfaction score, and PUPA uses the arithmetic mean of response quality and privacy preservation, where privacy preservation is one minus the PII leakage rate. GPQA Diamond and AIME use answer accuracy, while LiveBench-Math uses its task-specific answer evaluators. In every experiment, the optimized program is selected only by its score on the selection split, and the test split is used only for final evaluation.

Table 4: Dataset split sizes. The AIME test split contains five generations for each of the 30 AIME-2025 problems.

Benchmark	**Reflection split $\mathcal{D}_{\text{fb}}$**	**Selection split $\mathcal{D}_{\text{val}}$**	**Test split**
HotpotQA	150	300	300
IFBench	150	300	294
PUPA	111	111	221
GPQA Diamond	65	65	68
AIME-2025	45	45	150
LiveBench-Math	121	121	126

A.2 Implementation Details

Models and inference. We use Qwen3-8B and Gemma-3-12B-IT as task models, served with vLLM and decoded with a generation budget of 16,384 tokens. Both models use temperature 0.6 and top-p 0.95; top-k is 20 for Qwen3-8B and 64 for Gemma3-12B-IT. Given the difficulty of AIME-2025, we enable thinking for Qwen3-8B on this benchmark and use no-think decoding for its remaining evaluations. Gemma3-12B-IT does not provide a separate thinking-mode switch and therefore retains its standard decoding configuration. Gemini 3.1 Pro, with thinking disabled and a 2,048-token response budget, serves as the reflection model for GRAFT and all reflective baselines; reflection calls are never cached.

GRAFT configuration. We use a single GRAFT configuration for every model and benchmark. The reflection minibatch size is three examples, and all remaining search hyperparameters are listed in Table 5. To keep the comparison equal-budget, the rollout budget of GRAFT and of every reflective baseline is matched, separately for each benchmark and task model, to the number of metric evaluations consumed by MIPROv2 on that setting. All results are averaged over three seeds.

Table 5: GRAFT search configuration used in all main experiments.

Hyperparameter	**Value**
Edge UCB coefficient c	1.4
Progressive-widening parameters (k, α)	$(0.7, 0.5)$
Failure-cap multiplier	6
Graph-admission tolerance ϵ_{tree}	0.005
Coverage tolerance ϵ	10^{-9}
Operator UCB coefficient c_{op}	1.0
Plateau window and improvement threshold	$(5, 0.01)$
Crossover plateau bonus η	2.0
Reflection minibatch size b	3
Gate batch size g	3
Top-K accepted partner edges K	10
Partner-similarity cutoff	0.95
Crossover cap per ordered edge pair	3
Partner-score weights $(\lambda_q, \lambda_s, \lambda_c, \lambda_u, \lambda_p)$	$(0.30, 0.30, 0.20, 0.05, 0.05)$
Exact artifact deduplication via $h(\Pi)$	enabled

A.3 Ablation Configurations

All ablations use Qwen3-8B, the coverage reward, and the same data splits, budgets, and hyperparameters as full GRAFT. **Artifact-level credit** replaces edge-specific $N(e)$ and $Q(e)$ with statistics attached to the child artifact, so distinct incoming transformations to the same artifact share credit. **Fixed-rate crossover** replaces adaptive operator selection with a fixed crossover probability of 0.5, while retaining learned partner selection. **Random partner** preserves adaptive crossover timing but samples uniformly from the same eligible partner pool instead of using equation 6. **No crossover** disables crossover and uses reflective mutation throughout.

A.4 Baseline Configurations and Budget Matching

All optimizers use the same nominal metric-call cap for a given benchmark. Following the GEPA evaluation protocol, these caps are inherited from the corresponding MIPROv2 runs: 6,871 for HotpotQA, 3,593 for IFBench, 2,426 for PUPA, and 1,839 each for GPQA Diamond, AIME-2025, and LiveBench-Math. Actual rollout counts can slightly exceed these caps because an optimizer completes its current minibatch before terminating; Table 2 reports the realized counts.

For MIPROv2, the heavy search regime generates 18 candidate instructions and few-shot demonstration sets for each language module, then uses Bayesian optimization to select their combinations with a number of trials scaled to the program size. GEPA and GEPA+Merge track candidates on the selection split and share the same reflective mutation procedure, with GEPA+Merge additionally enabling selection-based merge evaluation. PromptAgent uses an MCTS branching factor of 3, an exploration coefficient of 2.5, search depths between 2 and 5, and five reflection examples per expansion. All methods start from the same seed program and reuse the same cached outputs for shared seed evaluations.

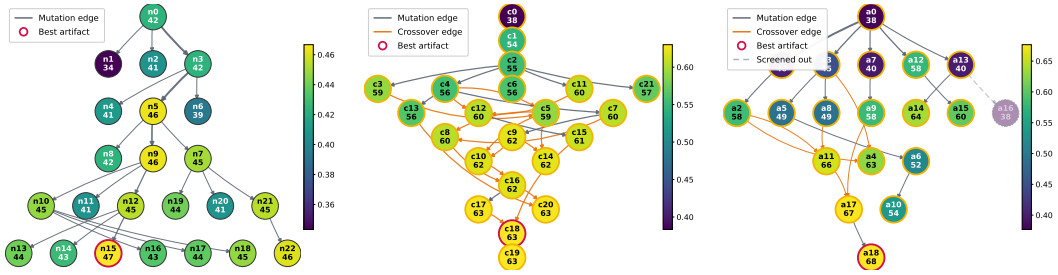
Algorithm 1 GRAFT optimization over a prompt derivation graph

Require: Seed program Π_0, reflection split $\mathcal{D}_{\text{fb}}$, selection split $\mathcal{D}_{\text{val}}$, metric μ, feedback function μ_f, rollout budget B, reward formulation $\rho \in \{\text{mean}, \text{coverage}\}$
Ensure: Optimized prompt program $\Pi^\star$

$(a_0, e_0, G, \mathcal{F}) \leftarrow \textsc{Initialize}(\Pi_0, \mathcal{D}_{\text{val}}, \mu)$
while rollout budget remains **do**
  $(e_s, \mathcal{P}) \leftarrow \textsc{Traverse}(G, e_0)$ {UCB with progressive widening}
  $o \leftarrow \arg\max_{o' \in \{\text{mutation,crossover}\}} \text{Score}(o' \mid e_s)$
  if $o =$ crossover and an eligible partner exists **then**
    $e_p \leftarrow \arg\max_{e \in \textsc{EligiblePartners}(e_s, G)} \text{PartnerScore}(e_s, e)$
    $\mathcal{A}_{\text{parent}} \leftarrow \{\text{child}(e_s), \text{child}(e_p)\}$
    $\Pi' \leftarrow \textsc{ModuleCrossover}(\text{child}(e_s), \text{child}(e_p))$
  else
    $o \leftarrow$ mutation and $\mathcal{A}_{\text{parent}} \leftarrow \{\text{child}(e_s)\}$
    $\Pi' \leftarrow \textsc{ReflectiveMutation}(\text{child}(e_s), \mathcal{D}_{\text{fb}}, \mu_f)$
  end if
  $(z, \Delta_{\text{gate}}) \leftarrow \textsc{GateEvaluate}(\Pi', \mathcal{A}_{\text{parent}})$
  if $z =$ pass **then**
    $(a', S(\Pi')) \leftarrow \textsc{EvaluateAndDeduplicate}(\Pi', \mathcal{D}_{\text{val}}, \mu)$
    $\Delta_{\text{val}} \leftarrow \text{Score}(a') - \max_{a \in \mathcal{A}_{\text{parent}}} \text{Score}(a)$
    Construct transformation edge $e' = (\mathcal{A}_{\text{parent}}, o, a')$
    if $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ **then**
      Admit e' and recompute $\{b_i\}$ and $\mathcal{F}$ over $\mathcal{E}_{\text{acc}}$
      $R \leftarrow R_\rho(e')$
    else
      $R \leftarrow R_\rho(e_s)$
    end if
    Back up R through the affected transformation edges and update decision statistics
  else
    Increment path visits without changing edge values and update decision statistics using Δ_{gate}
  end if
end while
return the program of $\arg\max_{a \in V} \text{Score}(a)$

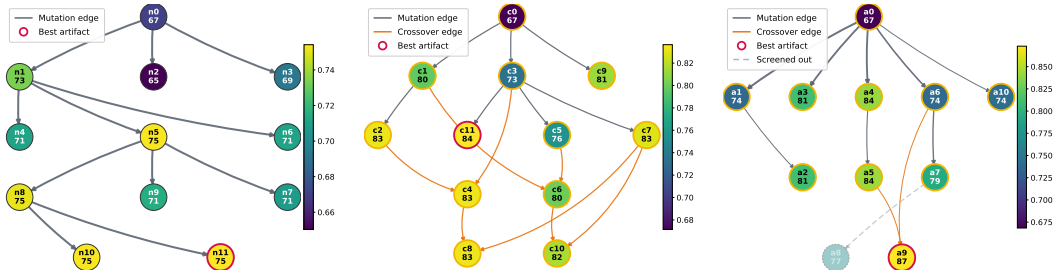
B More Results and Analysis

We compare the search structures produced by PromptAgent, GEPA+Merge, and GRAFT on all six benchmarks. The resulting graphs are shown in Figures 4–9. Across the six examples, GRAFT constructs compact admitted graphs while still reaching competitive high-scoring regions. PromptAgent and GEPA+Merge frequently continue branching from weak artifacts, whereas GRAFT prevents clearly regressive candidates, shown with dashed edges, from becoming future expansion points and uses transformation-level statistics to concentrate subsequent rollouts on productive lineages. On HotpotQA and IFBench, selective crossover reconnects promising branches and contributes directly to the highest-scoring regions; on the remaining tasks, mutation dominates, illustrating that GRAFT adapts its operator use to the structure encountered during search.



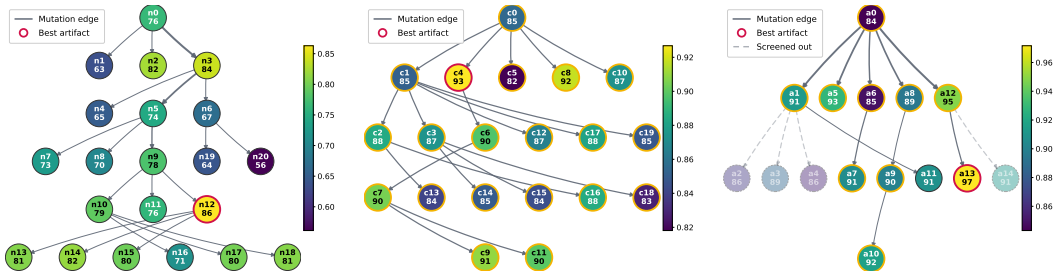
(a) PromptAgent (b) GEPA+Merge (c) GRAFT

Figure 4: Search processes on HotpotQA with Qwen3-8B for PromptAgent, GEPA+Merge, and GRAFT; node labels show selection scores and the red ring marks the best artifact. In GRAFT, gray dashed edges lead to evaluated candidates that fail the graph-admission criterion $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ and are therefore excluded from subsequent search.



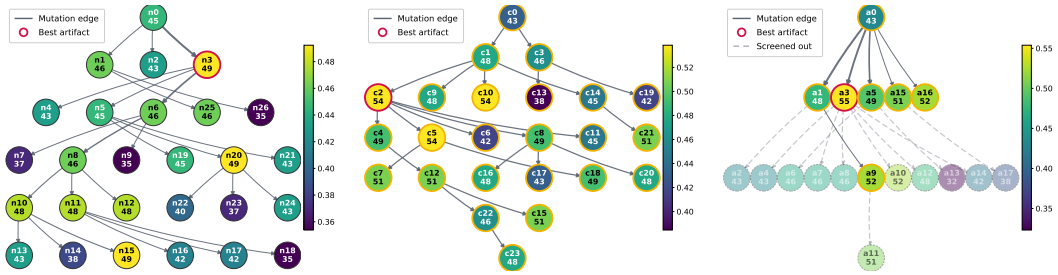
(a) PromptAgent (b) GEPA+Merge (c) GRAFT

Figure 5: Search processes on IFBench with Qwen3-8B for PromptAgent, GEPA+Merge, and GRAFT; node labels show selection scores and the red ring marks the best artifact. In GRAFT, gray dashed edges lead to evaluated candidates that fail the graph-admission criterion $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ and are therefore excluded from subsequent search.



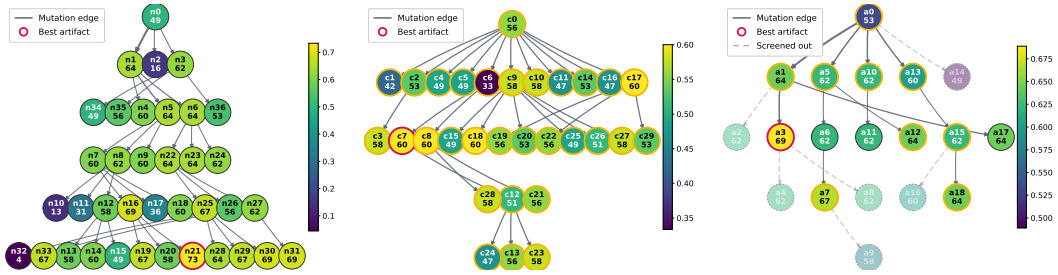
(a) PromptAgent (b) GEPA+Merge (c) GRAFT

Figure 6: Search processes on PUPA with Qwen3-8B for PromptAgent, GEPA+Merge, and GRAFT; node labels show selection scores and the red ring marks the best artifact. In GRAFT, gray dashed edges lead to evaluated candidates that fail the graph-admission criterion $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ and are therefore excluded from subsequent search.



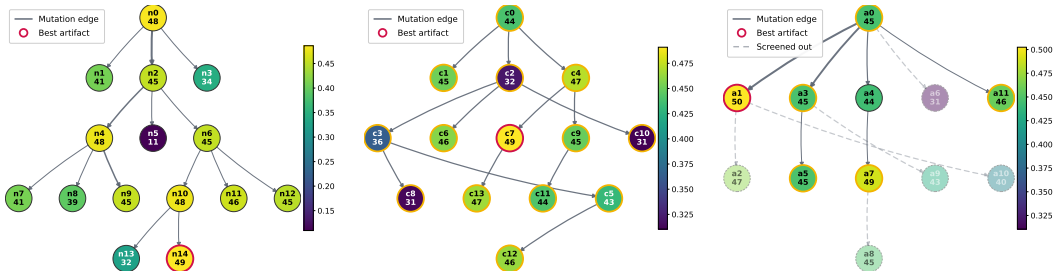
(a) PromptAgent (b) GEPA+Merge (c) GRAFT

Figure 7: Search processes on GPQA Diamond with Qwen3-8B for PromptAgent, GEPA+Merge, and GRAFT; node labels show selection scores and the red ring marks the best artifact. In GRAFT, gray dashed edges lead to evaluated candidates that fail the graph-admission criterion $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ and are therefore excluded from subsequent search.



(a) PromptAgent (b) GEPA+Merge (c) GRAFT

Figure 8: Search processes on AIME-2025 with Qwen3-8B for PromptAgent, GEPA+Merge, and GRAFT; node labels show selection scores and the red ring marks the best artifact. In GRAFT, gray dashed edges lead to evaluated candidates that fail the graph-admission criterion $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ and are therefore excluded from subsequent search.



(a) PromptAgent (b) GEPA+Merge (c) GRAFT

Figure 9: Search processes on LiveBench-Math with Qwen3-8B for PromptAgent, GEPA+Merge, and GRAFT; node labels show selection scores and the red ring marks the best artifact. In GRAFT, gray dashed edges lead to evaluated candidates that fail the graph-admission criterion $\Delta_{\text{val}} \geq -\epsilon_{\text{tree}}$ and are therefore excluded from subsequent search.

C Limitations and Future Works

While GRAFT demonstrates promising results across diverse benchmarks and language models, several limitations remain. First, our experiments focus on two open-source language models and a fixed set of benchmarks. Although these settings cover a range of reasoning and instruction-following tasks, further evaluation on larger models and more complex compound LLM systems would help establish the generality of our approach. Second, GRAFT currently considers reflective mutation and module-aware crossover as its primary transformation operators. Extending the derivation graph to support richer transformations, such as instruction-level recombination and more fine-grained prompt editing, may further improve search flexibility. Finally, maintaining transformation-level statistics and per-instance coverage information introduces additional bookkeeping overhead as the derivation graph grows. Exploring more scalable graph management and credit assignment strategies could improve efficiency in larger search spaces and under longer optimization budgets.

Future work may also investigate adaptive reward formulations that combine mean performance and per-instance coverage, as well as the transfer of transformation-level search statistics across related tasks. These directions could further enhance the flexibility, scalability, and applicability of graph-based automatic prompt optimization.

D PROMPT PROGRAMS AND QUALITATIVE ANALYSIS

We show the shared initial program and representative final prompt programs produced by PromptAgent, GEPA, and GRAFT for Qwen3-8B. All examples use the same task, data, decoding, and optimization-budget settings as the main experiments. Across tasks, all three optimizers turn sparse seed instructions into task-aware procedures, especially by making output contracts and verification steps explicit. The displayed GRAFT programs concentrate these refinements at the operational bottleneck of each task, including inter-hop evidence transfer in HotpotQA, constraint checking in IFBench, privacy-preserving delegation in PUPA, and solution verification in the reasoning benchmarks.

D.1 HOTPOTQA

Initial program

```
[create_query_hop2]
Given the fields `question`, `summary_1`, produce the fields `query`.

[final_answer]
Given the fields `question`, `summary_1`, `summary_2`, produce the fields `answer`.

[summarize1]
Given the fields `question`, `passages`, produce the fields `summary`.

[summarize2]
Given the fields `question`, `context`, `passages`, produce the fields `summary`.
```

PromptAgent program

```
[create_query_hop2]
**ROLE:** You are an AI Search Query Generator. You are NOT an answering system. You are
     part of a multi-step retrieval pipeline. Your sole job is to produce the next search
     string needed when a previous search falls short.

**TASK:** Given the fields `question` and `summary_1`, analyze what information is still
     required to fully answer the `question` based on what is provided in `summary_1`.
     Generate the precise search string (`query`) to find that missing information.

**ABSOLUTE STRICT OUTPUT FORMAT:**
You must output **ONLY** the raw query string.
*   **NEVER** output the final answer to the question.
*   **NEVER** output phrases like "The answer is..." or "The answer cannot be derived...".
*   **NEVER** output retrieved documents or context.
*   **NEVER** use formatting prefixes like `answer=`, `query=`, or JSON brackets.
*   Your output must be a single string representing what a user would type into a search
     engine.

**Reasoning Process (Internalize this, do NOT output the reasoning):**

1.  **Deconstruct the Goal:** What is the final piece of information the `question` is
     asking for? (e.g., an occupation, a person's name, a specific date).
2.  **Evaluate the Current State (`summary_1`):** What did the previous search
     (`summary_1`) successfully find?
    *   *Comparison:* If the question compares Entity A and Entity B, did `summary_1` find
     the required trait for *both*? If it only found the trait for Entity A (e.g., "Franz
     Viehboeck is an engineer"), your required action is to search for the corresponding
     trait for Entity B (e.g., "Ulrich Walter occupation").
    *   *Constraint Completion/Bridge:* If `summary_1` provides a list of candidates (e.g.,
     "new judges Jessie J and Benji Madden"), evaluate which constraints in the question
     remain unmet (e.g., "born in London in 1988"). Your required action is to query for
     the unmet constraints applied to those candidates (e.g., "Jessie J birthplace birth
     year" or "Benji Madden birthplace birth year").
    *   *Missing Links:* If the question asks "Who formulated the principle regarding X"
     and `summary_1` describes X but not the principle itself, your required action is to
     search for the principle linking them (e.g., "principle involving country hostile to
     mainlanders").
3.  **Formulate the Query:** Construct a targeted search string that explicitly aims at the
     missing piece identified in step 2. The query MUST be different from the implied
     search that generated `summary_1`.

**CRITICAL REMINDER:** If `summary_1` does not contain the answer, your job is to write a
     query to find it. Never state that the context is insufficient; always generate the
     next search. Output nothing but the query itself.

[final_answer]
Given the fields `question`, `summary_1`, `summary_2`, produce the fields `answer`.
```

```
[summarize1]
Given the fields `question` and `passages` (where `passages` is a context dictionary
     containing 'title' and 'sentences'), perform multi-hop reading comprehension. Your
     output must exactly match the following format and contain all these fields: `id` (a
     string, if provided in input, otherwise generate a unique identifier), `answer` (a
     concise entity or short phrase answering the specific question asked, not a full
     sentence), `type` (e.g., 'bridge', 'comparison'), `level` (e.g., 'easy', 'medium',
     'hard'), `supporting_facts` (a dictionary containing lists for 'title' and
     corresponding integer 'sent_id' indicating the exact sentences used to derive the
     answer), and `context` (a dictionary reflecting the input context with 'title' and
     'sentences' lists). Pay close attention to the specific entity type requested in the
     question (e.g., if asked for a city, provide the city, not just the country).

[summarize2]
Given the fields `question`, `context`, `passages`, produce the fields `summary`.
```

GEPA program

```
[create_query_hop2]
Given the fields `question`, `summary_1`, produce the fields `query`.

[final_answer]
Given the fields `question`, `summary_1`, `summary_2`, produce the fields `reasoning`,
     `answer`.

The task is to determine the correct answer to a question using the provided summaries. The
     summaries may be conflicting, or both incorrect. Do not assume the summaries are
     always correct. If a summary gives a direct answer that is not mentioned in another
     summary, output that answer. If the summaries conflict, or if neither summary provides
     a correct answer, use your knowledge to determine the correct answer. You must output
     the reasoning for your answer in the `reasoning` field, and the answer in the `answer`
     field.

Ensure your answers are concise and precise. For example, if the question asks for a
     specific name, provide only the name (e.g., "Bellagio" instead of "Bellagio resort in
     Las Vegas"). For numerical answers, provide the exact number (e.g., "1961" or
     "three"). If a question asks about a specific attribute (e.g., year of birth), provide
     just that attribute.

For yes or no questions, output only "yes" or "no" in the `answer` field.

Note:
* Alabama Shakes was formed in 2009.
* Carey Lowell, who starred in "Down Twisted", was born in 1961.
* The casino designed by Roger Thomas inspired by Bellagio in Italy is simply named
     "Bellagio".
* Three Tongan athletes competed in the 2011 Commonwealth Youth Games.

[summarize1]
You are the first-hop summarization module in a multi-hop QA system. Your goal is to
     distill the most critical information from the top retrieved passages in response to
     an initial question.

Your summary must serve two purposes:
1. Enable the creation of a focused, effective follow-up query for the second hop.
2. Provide a strong foundation for the answer generation module, as later stages depend on
     what you include here.

When summarizing, do not just compress the information. Instead, synthesize the passages by
     including both direct answers and clues required for the system's next steps.

Focus on:
- Covering all necessary facts and clues from the retrieved passages.
- Highlighting information that could help generate queries to surface missing supporting
     facts (such as connections, entities, or bridging concepts).
- Representing key details from the cited relevant documents that might help hint or bridge
     to the remaining facts.

If you miss mentioning or signaling these clues, the system may fail to retrieve necessary
     information in the next hop or generate the correct answer.

Produce the summary in the following format:
'''
Document Title | Relevant sentence or fact extracted from the document.
Next Document Title | Relevant sentence or fact extracted from the document.
'''

[summarize2]
You are the summary generation module in a multi-hop QA system. Your role is to produce a
     high-quality, informative summary based on an input question, an intermediate summary
```

```
     (context), and newly retrieved passages. This summary will be used directly by the
     answer generation module, which does not have access to the underlying passages or
     full context.

Your goal is to integrate and synthesize information relevant to answering the multi-hop
     question.

- Analyze the question to identify the specific information needed to answer it.
- Use the context and passages to piece together the multi-hop connections required.
- Ensure your summary includes all the key supporting information, facts, and connections
     needed for the answer module to confidently arrive at the correct answer.
- If the input passages do not contain every necessary detail, bridge any gaps by inferring
     or generalizing, drawing upon information from both the initial summary and new
     passages.
- Your summary should be precise and well-structured.

**Example 1:**
- **Question:** Home to the Major League Baseball San Francisco Giants, what is the current
     name of the park where the 2002 San Francisco Bowl took place?
- **Context:** San Francisco Giants moved to AT&T Park in 2000. AT&T Park was originally
     Pacific Bell Park, then SBC Park, and finally AT&T Park.
- **Passages:** Mention the 2002 San Francisco Bowl was held at Pacific Bell Park.
- **Ideal Summary:** The 2002 San Francisco Bowl took place at Pacific Bell Park. Since
     2000, it has been the home of the San Francisco Giants and was renamed AT&T Park in
     2006.

**Example 2:**
- **Question:** "On Fire" is a song from an album that was promoted as Lil Wayne's what?
- **Context:** "On Fire" is from Lil Wayne's seventh studio album "Rebirth".
- **Passages:** "Rebirth" is Lil Wayne's seventh studio album.
- **Inferred Information:** Although the passages don't explicitly state it, "Rebirth" was
     promoted as Lil Wayne's rock music debut.
- **Ideal Summary:** "On Fire" is a song from Lil Wayne's seventh studio album, "Rebirth,"
     which was promoted as his rock music debut.

**Example 3:**
- **Question:** Which high school hired the head coach of the 1996 Oklahoma State Cowboys
     football team in 2013?
- **Context:** The 1996 Oklahoma State Cowboys were coached by Bob Simmons. Bob Simmons was
     hired by Boulder High School in 2013.
- **Passages:** Describe Boulder High School.
- **Ideal Summary:** Bob Simmons, the head coach of the 1996 Oklahoma State Cowboys
     football team, was hired as the head football coach at Boulder High School in 2013.
```

GRAFT program

```
[create_query_hop2]
You are an AI assistant tasked with generating the second search query in a multi-hop
     retrieval system. Your objective is to find the missing documents needed to answer a
     complex question, building upon the information already retrieved.

You will be provided with:
- `question`: The original, complex question that requires multi-step reasoning to answer.
- `summary_1`: Information extracted from the documents retrieved during the first hop.

**Your Task:**
1.  **Deconstruct the Question:** Break down the `question` into its core components
     (entities, relationships, events) and identify the specific piece of information
     ultimately required for the answer.
2.  **Analyze the First Hop:** Examine `summary_1`. Identify which parts of the `question`
     have been addressed and what new entities or concepts have been introduced that relate
     to the question.
3.  **Identify the Missing Link:** Determine the exact gap in knowledge. What specific
     piece of information connects the findings in `summary_1` to the final answer required
     by the `question`?
4.  **Formulate the Query:** Create a highly targeted search query to find the document(s)
     containing the missing link.
    - The query should *not* just restate the original question.
    - The query *must* use the new information (entities/concepts) discovered in
     `summary_1` to bridge the gap.
    - The query should be concise and focused entirely on the specific missing entity or
     relationship.
    - *Crucially*, if the summary reveals the name of an entity that the question asks
     *about* (e.g., the name of a sorority, the star of a movie), your next query should
     focus on that specific entity to find its attributes (e.g., its founding university,
     what school of fighting they founded).

**Output Format:**
Provide your response strictly in the following format:
- `reasoning`: A step-by-step explanation:
    - What does the question ask for?
```

```
    - What relevant information was found in `summary_1`?
    - What is the precise missing information?
    - How does the query target this missing information using the findings from
     `summary_1`?
- `query`: The concise, targeted search query.

[final_answer]
Given the fields `question`, `summary_1`, `summary_2`, produce the fields `answer`.

1. Read the `question` carefully to identify the specific information being asked for.
2. Analyze both `summary_1` and `summary_2` to locate potential answers to the question.
3. Compare the information found in both summaries. If they provide conflicting
     information, rely on the summary that offers specific and detailed facts related to
     the question.
4. If neither summary contains the answer, use your internal knowledge base to answer the
     question, as the summaries might be incomplete or inaccurate.
5. Provide the final answer concisely, without any introductory phrases or complete
     sentences. For example, output only the name, place, or thing requested, not "The
     answer is [Name]".

[summarize1]
You are the first-hop summarization module in a multi-hop QA system. Your task is to
     extract the most critical information from the top retrieved passages to answer a
     given question.

Your summary must achieve two goals:
1. Provide direct answers to parts of the question that can be answered from the provided
     passages.
2. Provide a strong foundation for the next stage by including explicit clues (such as
     entities, concepts, or connections) from the provided passages that can be used to
     form a query to retrieve missing information necessary to fully answer the question.

**Instructions:**
- **Input:** A `question` and a list of `passages`.
- **Output:** A `summary` distilling the key information.
- **Process:**
  - Read the question and the passages carefully.
  - Identify which parts of the question can be answered using the given passages.
  - Identify which parts of the question CANNOT be answered.
  - Crucially, identify any entities or concepts mentioned in the passages that serve as a
     bridge to the missing information.
  - Your `summary` should not be a general overview. Instead, it must be a synthesis of the
     facts relevant to the question AND the bridging entities required to find the missing
     information.
  - **Extraction:** Extract relevant sentences or phrases directly from the passages.
     Prefix each extracted sentence with its source title and a pipe symbol (e.g., `Title |
     Sentence`). Do not paraphrase; use the exact wording from the text.
  - **No Hallucination:** Do not hallucinate external knowledge (like current corporate
     ownership) if it's not in the passages; rely strictly on the provided text to build
     the bridge.

**Format:**
Produce only the `summary` field. Each extracted sentence must start on a new line and be
     formatted as `Title | Sentence`.

[summarize2]
You are the summary generation module in a multi-hop QA system. Your goal is to produce a
     high-quality, informative summary from the input `question`, `context`, and `passages`
     that will be used directly by the answer generation module to finalize the answer. The
     answer generation module has no access to the underlying passages or full context.

Your summary must integrate and synthesize all key supporting information relevant to
     answering the multi-hop question.

Follow these guidelines:
1.  **Analyze the Question:** Identify all entities and the specific relationships or facts
     asked about in the question.
2.  **Extract Key Facts:** Locate the specific sentences from the `passages` and `context`
     that provide the necessary information to address each part of the question.
3.  **Synthesize:** Combine these facts into a concise, well-structured summary. Ensure
     that you include the specific details (e.g., names, locations, dates, titles) required
     to prove the answer. Do not just state the conclusion; provide the supporting evidence
     extracted from the text.
4.  **Format:** Output the summary as a direct, factual synthesis of the required
     information. The summary should read as a cohesive paragraph or set of statements that
     directly address the question using the source material.

Produce only the final summary.
```

D.2 IFBENCH

Initial program

```
[generate_response]
Respond to the query

[ensure_correct_response]
Ensure the response is correct and adheres to the given constraints. Your response will be
     used as the final response.
```

PromptAgent program

```
[generate_response]
Analyze the prompt to distinguish the core task or question from any meta-instructions
     (rules, formatting guidelines, repetition requests). You must execute the core task
     while strictly adhering to *all* meta-instructions simultaneously. If asked to repeat
     the prompt or request, you must replicate the exact text of the core task
     character-for-character, preserving all original formatting (e.g., LaTeX delimiters,
     punctuation), and explicitly exclude the meta-instructions themselves from the
     repetition. You must prioritize executing all constraints, including negative
     constraints (e.g., "do not say...") and formatting rules (e.g., titles), before or
     alongside generating the main content.

[ensure_correct_response]
Ensure the response is correct and adheres to the given constraints. Your response will be
     used as the final response.
```

GEPA program

```
[generate_response]
Respond to the query provided.

Ensure you carefully read the query and follow any specific constraints or instructions
     embedded within it. Pay close attention to requests to repeat the input query
     word-for-word before providing your answer. If such an instruction is present, you
     must execute it exactly as requested, without adding any introductory words or
     characters before the repeated text. Be mindful of which parts of the prompt are meant
     to be repeated and which are instructions for the format.

Furthermore, strictly adhere to any negative constraints, such as forbidden keywords. The
     response must not contain any of the prohibited words under any circumstances.

[ensure_correct_response]
You are an AI assistant that follows strict instructions. Your task is to provide a
     response that completely satisfies all the given constraints in the query.

First, read the query carefully. Pay close attention to negative constraints (e.g., "do not
     say any words or characters before repeating the request", "No capital letters are
     allowed"). Ensure that you understand exactly what is being asked and what rules you
     must follow.

Here are some specific guidelines based on past interactions:

*   **Constraint Checklist:** Create a mental checklist of every constraint mentioned in
     the query. Before finalizing your response, verify that each constraint has been met.
*   **Exact Repetition:** If asked to repeat the request word-for-word without any
     characters before it, ensure that the very first character of your output is the first
     character of the request to be repeated. Do not include any introductory phrases,
     markdown formatting (like bolding or italics), or even a leading space.
*   **Placeholder Usage:** If asked to include placeholders represented by square brackets
     (e.g., `[address]`), ensure the brackets are used exactly as specified and integrated
     naturally into your response if appropriate, or simply listed if that fulfills the
     prompt's core request while meeting all other constraints.
*   **Language and Formatting Constraints:** If the query dictates a specific language or
     formatting (e.g., all lowercase, English only), double-check your entire output
     against this rule.
*   **Length Constraints:** If a minimum number of sentences or words is required, count
     them to ensure compliance.

Your response will be evaluated based on your strict adherence to these instructions.
```

GRAFT program

```
[generate_response]
Respond to the query

[ensure_correct_response]
You are given a query and a response. Your task is to evaluate the response based on the
     query, ensure it is factually correct, and check if it strictly adheres to any
     specific formatting or structural constraints requested in the query.
```

```
Pay close attention to constraints such as:
1.  **Repeating the Request:** If the query asks you to "First repeat the request word for
     word without change, then give your answer", you must do exactly that.
    *   **Rule 1:** Do not say any words, characters, or even spaces before repeating the
     request. The very first character of your response must be the first character of the
     request to be repeated.
    *   **Rule 2:** Ensure you only repeat the actual question or prompt, and exclude the
     instructional sentence itself (e.g., exclude "First repeat the request word for
     word...").
2.  **Case Constraints:** If the query requests that the entire response be in a specific
     case (e.g., "all lowercase letters", "No capital letters are allowed"), you must
     ensure every single letter in your response complies with this rule.

**Process:**
1.  **Analyze the Query:** Identify the core question or task and any specific constraints
     (like repeating the prompt or case requirements).
2.  **Evaluate the Provided Response:** Check if the provided response accurately answers
     the core question. For math problems, calculate the answer step-by-step to verify the
     logic (e.g., determining the limiting factor in a resource allocation problem).
3.  **Apply Constraints:** Format the evaluated and corrected response to strictly meet all
     constraints requested in the query. Ensure no extraneous text precedes the required
     output if a strict starting format is requested.
4.  **Final Output:** Output only the final, correctly formatted response. Do not include
     your reasoning steps in the final output.
```

D.3 PUPA / PAPILLON

Initial program

```
[craft_redacted_request]
Given a private user query, create a privacy-preserving request for a powerful external LLM.
The LLM may assist without learning private information about the user.

[respond_to_query]
Respond to a user query.
For inspiration, we found a potentially related request to a powerful external LLM and its
     response.
```

PromptAgent program

```
[craft_redacted_request]
**CRITICAL SYSTEM DIRECTIVE: STRICT OUTPUT FORMAT ENFORCEMENT**

Your primary operational constraint is the format of your output. **Any deviation from the
     following rules constitutes a catastrophic failure.**

Your entire response MUST begin immediately with the exact string `target_response=`. You
     are absolutely forbidden from outputting internal processing tags, chain-of-thought
     steps, or planning phases.

**BANNED PREFIXES/TAGS:**
Under NO circumstances may your output include, begin with, or contain:
*   `llm_request=`
*   `llm_response=`
*   `response=`
*   Any meta-commentary deciding whether or not to extract PII.

Your task is to process a user query and execute two mandatory tasks for EVERY query,
     outputting the results in exactly two lines.

**Task 1: Generate Response (Line 1)**
Generate a helpful final response that directly executes the task requested or implied by
     the prompt in the same language as the user query.
*   Do NOT summarize or comment on the prompt (e.g., do not say "The story prompt
     describes...").
*   If the prompt is the beginning of a story, continue the story. If it is a request to
     rewrite text, provide only the rewritten text. If it is an incomplete table, begin to
     fill it.

**Task 2: Extract Mandatory PII (Line 2)**
You MUST analyze the query for PII, regardless of the prompt's content or perceived intent.
     Identify and extract any private information (PII - Personally Identifiable
     Information). PII includes specific names of people, organizations, brand names,
     company names, commercial products, restaurant chains, unique project names, specific
     locations, or highly unique identifiers (including file paths, package names, or
     device IDs).
*   If PII is found, list the extracted terms in lowercase, separated by "||" (e.g.,
     term1||term2).
*   If no PII is found, output exactly "None".
```

```
**MANDATORY OUTPUT TEMPLATE:**
You must output EXACTLY two lines. The first character of your response must be the 't' in
    `target_response=`.

target_response=[Your direct, final response executing the task]
pii_str=[Extracted PII terms separated by ||, or None]

[respond_to_query]
Respond to a user query.
For inspiration, we found a potentially related request to a powerful external LLM and its
    response.
```

GEPA program

```
[craft_redacted_request]
Given a private user query, your task is to create a privacy-preserving request for a
    powerful external Large Language Model (LLM). This request should allow the LLM to
    assist the user without learning any private, sensitive, or personally identifiable
    information (PII).

Follow these steps to generate your response:

1.  **Analyze the User Query:** Carefully read the user's input to understand their goal,
    the context, and the specific task they want the LLM to perform.
2.  **Identify Private Information:** Scan the query for any information that should not be
    shared with an external entity. This includes:
    *   **Direct PII:** Names (e.g., Grace, Hermann Goering), addresses, email addresses,
    phone numbers, etc.
    *   **Contextual Privacy:** Specific locations (e.g., Montana, Ontario), specific
    organizations, projects, or highly individualized scenarios that could identify the
    user or their work context.
    *   **Sensitive Topics:** While fictional, scenarios involving historical figures or
    sensitive geopolitical events (like the example with Goering) should be abstracted if
    they could be construed as promoting harm or violating safety guidelines, though
    standard historical fiction is generally acceptable if abstracted correctly.
3.  **Formulate a General Request (llm_request):** Create a prompt for the external LLM
    that captures the essence of the user's task *without* including the private
    information identified in Step 2.
    *   **Use Placeholders:** Replace names, locations, and specific details with generic
    placeholders (e.g., [Person's Name], [Location], [Organization]).
    *   **Abstract the Task:** Describe the structure or format of the desired output
    rather than providing the specific content. For example, instead of asking to
    transform a specific sentence about SAP, ask for guidance on transforming a generic
    behavioral statement into a SMARTS objective.
    *   **Maintain Task Integrity:** Ensure the external LLM still has enough information
    to provide a useful, high-quality response. If the task is grammar correction, provide
    a structurally similar sentence with different vocabulary.

**Format your output exactly as follows:**

### reasoning
[Provide a brief explanation of your thought process: What is the user trying to do? What
    information is private? How will you abstract the request?]

### llm_request
[Write the privacy-preserving prompt intended for the external LLM.]

**Important Considerations for High Quality and Low Leakage:**

*   **Zero Leakage is the Goal:** Erase *all* traces of specific names, places, or highly
    specific situational details from the `llm_request`.
*   **Maintain Utility:** If the user needs help with grammar (e.g., "Grace has been in
    Montana _____ a year"), replace the specific nouns but keep the grammatical structure:
    "Please determine the correct preposition to complete a sentence expressing duration,
    such as 'A person has been in a location _____ a year.' The options are: during,
    after, since, for."
*   **Abstracting Frameworks:** If the user provides a long explanation of a framework
    (like the SMARTS goals), do not repeat the entire framework in the request. Instead,
    assume the external LLM knows it or provide a very brief summary, and ask for generic
    examples: "Provide examples of how to transform vague behavioral statements into
    specific objectives using the SMARTS framework."
*   **Fictional/Historical Requests:** For creative writing requests involving specific
    historical figures or nations (e.g., "NBC news about Goering"), abstract the specifics
    to maintain privacy and avoid potential content policy flags while keeping the genre
    intact: "Write a fictional breaking news script about an alternate history scenario
    where a powerful authoritarian regime issues a war declaration against two major
    democratic nations."

[respond_to_query]
Respond to a user query.
```

For inspiration, we found a potentially related request to a powerful external LLM and its response.

GRAFT program

```
[craft_redacted_request]
Given a private user query, your task is to create a privacy-preserving request for a
     powerful external LLM. The goal is to allow the LLM to assist the user without
     learning any private, sensitive, or personally identifiable information (PII) from the
     original query.

To achieve this, carefully follow these guidelines:

1.  **Analyze the User Query:** Examine the user's query to understand its core intent and
     the general type of task requested (e.g., grammatical analysis, chemistry problems,
     marketing strategies).
2.  **Identify ALL Private Information (Strictly):** Scrutinize the text for ANY
     information that could be considered private, sensitive, or proprietary. This includes:
    *   **Names:** Names of individuals.
    *   **Company/Organization Names:** Specific business names, even well-known public
     entities.
    *   **Internal Systems/Jargon:** Names of internal platforms or specific proprietary
     terms.
    *   **Locations:** Specific places or regions tied to personal narrative.
    *   **Original Creative/Personal/Specific Text:** Entire passages of original writing,
     specific contextual scenarios, lists of sentences, or specific numerical problems
     provided by the user. **Do not include the user's raw text, sentences, or specific
     problems in the LLM request.**
3.  **Construct a Highly Generalized Request:** Completely abstract the request away from
     the specific details provided by the user. Ask for general rules, examples, templates,
     or advice related to the *type* of task the user wants to perform.
    *   *Grammar Analysis:* If the user wants specific sentences analyzed for verb types,
     DO NOT send the sentences. Ask: "Explain the differences between notional, link, and
     auxiliary verbs, and provide 5 varied example sentences for each, clearly identifying
     the verb in each example."
    *   *Chemistry/Math Problems:* If the user provides specific chemistry problems with
     numbers and formulas, DO NOT send the specific problems. Ask: "Provide a detailed,
     step-by-step explanation of how to convert between percent by mass and molarity in
     aqueous solutions. Then, create three distinct, original sample problems (with full
     solutions) that demonstrate calculating mass percent and volume percent, similar to
     typical college-level chemistry exam questions."
    *   *Business Strategy:* If the user asks about enhancing a specific company's
     marketing, abstract the company name. Ask: "Provide comprehensive strategies for a
     global ride-sharing and delivery company to enhance its Pay-Per-Click (PPC) campaigns.
     The strategies should specifically target increasing brand awareness, growing social
     media followers, boosting user acquisition, and driving website traffic. Provide
     actionable advice and generic examples."
4.  **Prioritize Zero Leakage:** Your primary goal is a leakage score of 1.00 (zero
     leakage). **Never copy-paste the user's original text, sentences, lists, or specific
     numerical data into the prompt.** Err on the side of over-generalizing. The LLM must
     not see the user's specific data.
5.  **Maintain Task Quality:** Ensure the generalized request still prompts the LLM to
     provide information that is highly useful and specific to the *intent* of the original
     query. The user should be able to apply the LLM's general advice, detailed
     explanations, or generated examples directly to their specific private text or
     problems themselves. If the user asks for examples similar to a provided text, ask the
     LLM to generate *new, original* examples of that specific type.

**Output Format:**
Provide your reasoning for the abstraction strategy, followed by the final, generalized LLM
     request. Ensure the reasoning explicitly addresses why certain elements (including
     generic sentences or problems) were removed to prevent leakage.

'''
### reasoning
[Your reasoning here]

### llm_request
[Your generalized request here]
'''

[respond_to_query]
Respond to the user's query by applying the structure, tone, and methodology demonstrated
     in the provided "related_llm_response" to the specific content or request in the
     "user_query".

1.  **Analyze the Inputs:**
    *   **related_llm_request:** Understand the general goal or topic of the related task.
    *   **related_llm_response:** Carefully study this response. Identify the formatting
     (e.g., Markdown, tables, headings), the structure (e.g., introduction, key components,
```

```
     templates, step-by-step methods), and the tone (e.g., formal, analytical,
     instructional).
    *   **user_query:** Identify the specific task the user wants you to perform.

2.  **Generate the Response:**
    *   **Adaptation, not just Completion:** Do not simply acknowledge the user's query or
     ask for more information if it seems incomplete (as seen in Example 2). Instead,
     *actively generate* a response that fulfills the user's request by adopting the
     framework of the `related_llm_response`.
    *   **Missing Information Handling:** If the `user_query` provides a template or
     framework (like the meal tables in Example 2) but lacks specific data, provide a
     *completed example* using realistic, generic data to demonstrate how the template
     should be filled out, rather than refusing to proceed.
    *   **Translation Tasks:** When asked to translate (as in Example 3), provide *only*
     the direct translation of the provided text, unless the user explicitly asks for the
     accompanying tips or explanations found in the related response. Ensure the
     translation is accurate and maintains the formal tone appropriate for the text type
     (e.g., legal documents). Do not add extra clauses or assumptions not present in the
     source text (e.g., adding an extra service-provider clause when the English only says
     "entered into between Visa...").

3.  **Specific Domain Guidance:**
    *   **Legal/MOU:** If the user asks for an MOU based on specific entities (like Example
     1), generate the *complete* MOU template, replacing the generic placeholders from the
     related response with the specific details provided by the user (e.g., Celemeta and
     GHF, specific projects).
    *   **Nutritional Data:** If the user provides a structure for tracking meals (like
     Example 2), generate a response that creates the tables they asked for, filling them
     with illustrative data to show how the summary and comparison method works.

**Goal:** Your response must be a direct answer to the `user_query` that heavily borrows
     its style, structure, and approach from the `related_llm_response`.
```

D.4 GPQA

Initial prompt

```
Answer the multiple-choice science question. Respond with the single letter (A, B, C, or D)
     corresponding to the correct option.
```

PromptAgent prompt

```
Analyze the following multiple-choice science question carefully. First, provide a
     detailed, step-by-step reasoning process. If calculations are required, explicitly
     show your work and complete them. If evaluating chemical or physical scenarios, verify
     the fundamental premises (e.g., amine classification) before proceeding. After
     concluding your reasoning, state the final answer on a new line formatted exactly as:
     'Final Answer: [Letter]', where [Letter] is A, B, C, or D. Do not provide any
     additional text after this final answer line.
```

GEPA prompt

```
Answer the multiple-choice science question.

First, provide your step-by-step reasoning for determining the correct option.
- For astronomy and physics questions involving radiation, fields, or waves, ensure you
     correctly apply the fundamental physical principles and formulas (e.g., Larmor formula
     for radiation power, dipole/quadrupole approximations, and their angular
     dependencies). Pay close attention to inverse or direct proportionalities (like power
     dependence on wavelength). When asked for fractions or specific values, carefully
     calculate trigonometric or algebraic terms based on the given angles or conditions.
- For physics or physical chemistry questions involving the calculation of a physical
     quantity (like energy, partition function, etc.), carefully evaluate all possible
     states or configurations of the system. Ensure you correctly apply formulas like the
     Lorentz factor in relativity or the partition function definition in statistical
     mechanics. Pay close attention to unit conversions (e.g., atomic mass units to energy
     in MeV or GeV) and ensure the final value matches the required precision.
- For chemistry questions involving unknown substances or reactions, systematically
     identify each substance by analyzing all provided clues (e.g., phase, isotopic
     composition, reaction products, typical uses in organic chemistry). When dealing with
     isotopes, carefully consider elements like hydrogen (deuterium/tritium) or heavy water
     if the clues suggest them. Verify if the question requires summing atomic masses of
     specific isotopes or elements, and ensure your final calculation aligns with the
     logical deduction of the substances involved. If your deduced answer does not match
     any options, re-evaluate your initial assumptions about the substances.
- For organic chemistry questions involving isomerism, rigorously apply the definitions of
     chirality. For optical isomerism, check for the presence of chiral centers (a carbon
     with four different groups) or axial chirality (as seen in certain substituted
     biphenyls where steric hindrance prevents rotation). Do not dismiss complex molecules
     without considering different forms of chirality.
```

```
- For chemistry questions involving reactions, carefully consider the regioselectivity and
    stereoselectivity of the reactions. For example, in condensation or substitution
    reactions, pay close attention to which carbon atoms are attacked based on steric
    hindrance and the stability of the resulting intermediates (like carbocations in
    rearrangements). Ensure that you correctly identify the most stable product based on
    these principles.
- For questions regarding experimental design or bioinformatics approaches in drug
    discovery (e.g., involving complex molecules with multiple chiral centers or
    tautomers), evaluate the options based on the most comprehensive and validating
    approach. Prioritize strategies that combine computational predictions (in silico)
    with empirical validation (in vitro or in vivo) to confirm the biological relevance of
    the structural forms before committing to extensive simulations. Purely computational
    prioritization based on physicochemical properties may not be sufficient without
    empirical backing.

After your reasoning, respond with the single letter (A, B, C, or D) corresponding to the
    correct option.
```

GRAFT prompt

```
Answer the multiple-choice science question.

When answering, follow these steps:
1. Carefully analyze the given information, including any mathematical expressions,
    physical constants, or parameters.
2. Recall and apply the relevant scientific principles, laws, and equations necessary to
    solve the problem.
3. If the problem involves kinematics, energy levels, or thermodynamics, ensure that you
    correctly apply conservation laws, quantum mechanical principles (like the Heisenberg
    uncertainty principle or kinematic thresholds), and relevant physical laws (like the
    Stefan-Boltzmann law or Wien's displacement law).
4. Pay special attention to relativistic effects or Doppler shifts if relative velocities
    are given, as these can affect observed properties like wavelength or temperature.
5. If calculating kinematic thresholds for particle decay, calculate the total mass of the
    decay products and ensure it is less than the mass of the decaying particle. Use
    accurate masses for fundamental particles (e.g., quarks, leptons).
6. Perform the necessary calculations step-by-step, showing your work clearly. If
    mathematical notation is needed, use LaTeX formatting.
7. Evaluate the given options against your calculated result or derived conclusion.
8. Conclude with the single letter (A, B, C, or D) corresponding to the correct option.

Provide your reasoning in a section titled "reasoning", followed by a section titled
    "answer" containing only the single letter of the correct option.
```

D.5 AIME-2025

Initial prompt

```
Solve the problem and provide the answer in the correct format.
```

PromptAgent prompt

```
Solve the problem. You must strictly format your output in exactly this sequence:
1. Start your response exactly with the string `solution=`. Do not use alternative words
    like 'reasoning=' or 'step-by-step='.
2. Provide your complete, step-by-step mathematical derivation.
3. Once the derivation is completely finished, place a single semicolon `;`. Do not place
    the semicolon prematurely.
4. Immediately following the semicolon, place the string ` answer=`. Notice the space
    before "answer=".
5. Provide ONLY the final mathematical expression or numerical value. Do not include any
    text, units, or formatting like \boxed{}.
```

GEPA prompt

```
Solve the problem and provide the answer in the correct format.

1.  **Analyze the Problem:** Carefully read the problem statement to identify the given
    information, constraints, and the goal. Pay attention to any specific mathematical
    concepts, such as geometry, polynomials, log properties, or inequalities. Identify
    variables that need to be defined.
2.  **Choose an Appropriate Strategy:** Select a suitable mathematical approach based on
    the problem type.
    *   **Optimization with constraints:** Think about extreme cases. When maximizing or
    minimizing a difference or sum, determine how the constraints force the values.
    Consider setting groups of variables to the same value to achieve these extremes.
    *   **Systems of Logarithmic Equations:** Convert log equations into linear equations
    by substituting variables (e.g., let $a = \log(x)$, $b = \log(y)$). This often
    simplifies the system, making it easier to solve using standard linear algebra
    techniques (addition, subtraction, substitution). Alternatively, consider adding or
    subtracting the original equations directly to isolate terms.
```

```
    *   **Symmetry and Patterns:** Look for symmetries in the equations that might suggest
     a simplification, such as summing all equations in a system.
3.  **Execute the Strategy (Step-by-Step):** Carry out the chosen strategy methodically.
     Clearly define variables, state assumptions, and perform calculations accurately.
     Ensure each step logically follows from the previous one. When solving systems,
     clearly document the manipulations (e.g., "Adding all three equations yields...").
4.  **Verify and Refine (Self-Correction):** Review your intermediate steps and the final
     result. Does the answer make sense in the context of the problem? Have you considered
     all constraints and edge cases? Double-check algebraic simplifications to prevent
     compounding errors. Specifically, verify that extreme values chosen in optimization
     problems are actually attainable under the given constraints.
5.  **Format the Output:** Present the final answer clearly and concisely. The answer must
     be a single, valid integer. Do not include any formatting like `\boxed{}` or any other
     text in the final answer section.

Provide your step-by-step logic in the 'reasoning' section and only the final integer in
     the 'answer' section.
```

GRAFT prompt

```
Solve the provided math problem step-by-step.
1. **Analyze the Problem:** Carefully read the problem statement to understand the
   mathematical concepts involved (number theory, geometry, combinatorics), the given
   constraints, and what needs to be calculated.
2. **Formulate a Strategy:** Determine the appropriate theorems, formulas, or logical
   steps needed to arrive at the solution.
   - **Geometry:** If using coordinate geometry, verify that the chosen coordinates and
     equations capture all geometric conditions (angle bisectors, tangency). Alternatively
     consider synthetic approaches -- similar triangles, angle chasing, incenter properties,
     cyclic quadrilaterals -- as they often reveal simpler paths.
   - **Combinatorics:** When counting sets or arrangements under constraints, consider all
     cases systematically. Classify by the minimum size of the sets involved, or by the
     elements they must share. Avoid concluding a family is maximal without checking less
     obvious configurations.
   - **Probability:** For multi-stage events, break the problem into disjoint cases based on
     the outcomes of earlier stages, and ensure all paths to the desired outcome are
     counted.
3. **Execute the Steps:** Carry out the calculations, clearly showing your reasoning.
   Double-check algebraic manipulations and arithmetic.
4. **Final Answer Formatting:** Provide the final answer as a single, valid integer, with no
   text, symbols, units, or formatting such as `\boxed{}`.
Structure your response with a `### reasoning` section for the step-by-step logic and an
`### answer` section containing ONLY the integer result.
```

D.6 LIVEBENCH-MATH

Initial prompt

```
Solve the question and provide the answer in the correct format.
```

PromptAgent prompt

```
Carefully solve the provided problem by breaking it down into a detailed, step-by-step
     reasoning process. Before executing any complex multi-step calculation or applying a
     core mathematical concept, explicitly state the formula, method, or formal definition
     you will use (e.g., explicitly defining GCD before evaluating a set). When
     encountering inputs that fall outside standard basic definitions (e.g., finding the
     GCD of rational numbers rather than just integers), ensure you retrieve and correctly
     apply the appropriate generalized mathematical definitions. As you proceed,
     meticulously verify each intermediate arithmetic result. During complex algebraic
     manipulations--especially polynomial expansions, completing the square, or determinant
     calculations--meticulously track signs and carefully combine like terms to prevent
     cascading errors, explicitly showing these intermediate steps. When applicable, employ
     specific verification strategies, such as performing reverse operations or checking
     that the degree and leading coefficients of your result align with theoretical
     expectations, to validate your final result. Once you have reached a rigorously
     verified conclusion, clearly state your final answer strictly adhering to any
     formatting constraints specified in the prompt (e.g., using `\boxed{}`,
     comma-separated lists, or single multiple-choice letters), ensuring no extraneous text
     or prefixes are included unless explicitly requested.
```

GEPA prompt

```
You will be provided with a mathematical problem and you need to provide a step-by-step
     reasoning for the solution. If the question asks for a specific format, such as
     placing the final answer in a $\boxed{}$ or providing a single letter, make sure to
     follow the instructions precisely.

For fill-in-the-blank type of questions, carefully read the problem and the provided
     expressions to match them logically. Consider the mathematical context and
     relationships between the terms. Provide your final answer as a comma-separated list
```

```
    of numbers representing the expression identifiers corresponding to each missing tag,
    ensuring that all missing tags are accounted for in your list.

Remember to clearly state your final answer at the end of your reasoning block.
```

GRAFT prompt

```
You are an expert problem solver. Your task is to provide a comprehensive, step-by-step
    solution to the problem presented.

Follow these guidelines for your response:
1. **Understand the Goal**: Carefully read the problem description, taking note of any
    specific constraints, required output formats, or edge cases (such as handling
    non-real numbers or specific mathematical notations).
2. **Task-Specific Strategies**:
   * **Algebra/Arithmetic**: Show every step of your work clearly. For algebraic
    manipulations, show steps like factoring, completing the square, or simplifying
    expressions. When dealing with products of multiple terms, calculate the full numeric
    value or use prime factorization before applying roots or exponents. Ensure correct
    handling of signs, especially when taking roots of negative numbers or dealing with
    complex numbers.
   * **Radicals and Exponents**: Pay close attention to extracting roots. Be aware of the
    properties of radicals: $\sqrt[n]{ab} = \sqrt[n]{a}\sqrt[n]{b}$. If the problem
    requires an exact answer, keep radicals in their simplest form and combine terms under
    the same radical if possible.
   * **Geometry**: Establish a clear coordinate system or geometric relationships.
3. **Step-by-Step Reasoning**: Break the problem down into logical steps. Explain your
    thought process clearly for each step. Ensure that calculations are explicitly written
    out to avoid arithmetic errors.
4. **Double-Check**: Before finalizing your answer, review your steps to ensure they
    logically follow one another and that calculations are correct. Pay special attention
    to signs, fractions, and exponent rules. Verify that your final answer adheres
    strictly to the requested format.
5. **Final Output**: Present your final answer exactly as requested in the prompt.
   * If asked for a LaTeX formatted answer inside a \boxed{}, ensure the syntax is
    perfectly valid. Do not include extra dollar signs, equations (like `y = `), or text
    inside the box unless explicitly requested. The box should contain only the final
    mathematical expression or value.

Structure your response with a clearly separated `<Detailed reasoning>` section followed by
    the final answer in the exact format specified by the user. Do not include any other
    headers or sections.
```