FUSION IS THE NEW MUTATION: BANDIT-GUIDED EVOLUTION ON WORKFLOW GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated agentic workflow optimization relies on costly evaluations, making it essential to allocate a limited evaluation budget effectively. Multi-parent fusion can reuse designs from previously discovered workflows, but identifying promising parent combinations requires learning from limited fusion feedback. We introduce **DAGO** (Directed Acyclic Graph Optimization), a contextual-bandit-guided framework that learns which parent workflows to fuse under a limited evaluation budget. DAGO formulates each candidate parent combination as an **arm**, represented by pretrained embeddings of its constituent workflows' code and prompts. A diagonal LinUCB policy learns a shared reward model across arms and balances exploitation of arms with high predicted offspring quality against uncertainty-driven exploration. After an arm is selected, an LLM generates a child workflow through summary-guided fusion, and the child's validation score serves as the reward for updating the bandit. A shared directed acyclic graph maintains discovered workflows and their multi-parent lineage, providing an expanding pool of parents for subsequent arm proposals. Across six benchmarks covering mathematical reasoning, code generation, and question answering, DAGO achieves the highest macro-average score among the evaluated baselines. Under matched validation-evaluation budgets, it improves over AFlow from 80.3 to 81.7 while reducing aggregate search expenditure by 11.2%. Ablation studies show that LinUCB-guided arm selection outperforms both random selection and its exploration-free variant, supporting the value of feedback-driven selection and exploration–exploitation balance.

1 INTRODUCTION

Agentic workflows coordinate large language model (LLM) calls, tools, and control-flow logic to solve complex tasks. Automating their design turns workflow construction into a search problem over executable programs. ADAS generates new agentic systems by conditioning on an archive of previous discoveries (Hu et al., 2025), while AFlow explores workflow code through a variant of Monte Carlo Tree Search (Zhang et al., 2025d). In these settings, assessing a candidate requires executing its LLM calls and tools on validation examples, making search evaluations costly. Under a limited evaluation budget, an optimizer must therefore decide not only how to generate new workflows, but also which generation decisions are worth evaluating.

Multi-parent fusion offers a way to reuse useful designs from previously discovered workflows. For example, a workflow with effective problem decomposition may provide useful guidance for improving another workflow with a strong verification procedure. Existing approaches already explore workflow recombination: EvoFlow uses LLM-based crossover (Zhang et al., 2025a), and MermaidFlow introduces constraint-preserving crossover over workflow graphs (Zheng et al., 2025). However, an effective fusion operator does not by itself determine which parents should be fused. High individual validation scores do not guarantee that a parent combination will produce a strong child, and the number of possible combinations grows combinatorially as more workflows are discovered. Evaluating every combination is impractical, while selecting parent combinations at random does not use past fusion outcomes to guide future choices. The central challenge is thus to learn which combinations merit the next expensive fusion and evaluation.

This motivates our research question: *Under a limited evaluation budget, how can we learn from past fusion outcomes to select promising parent workflow combinations?* We introduce **DAGO**

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.

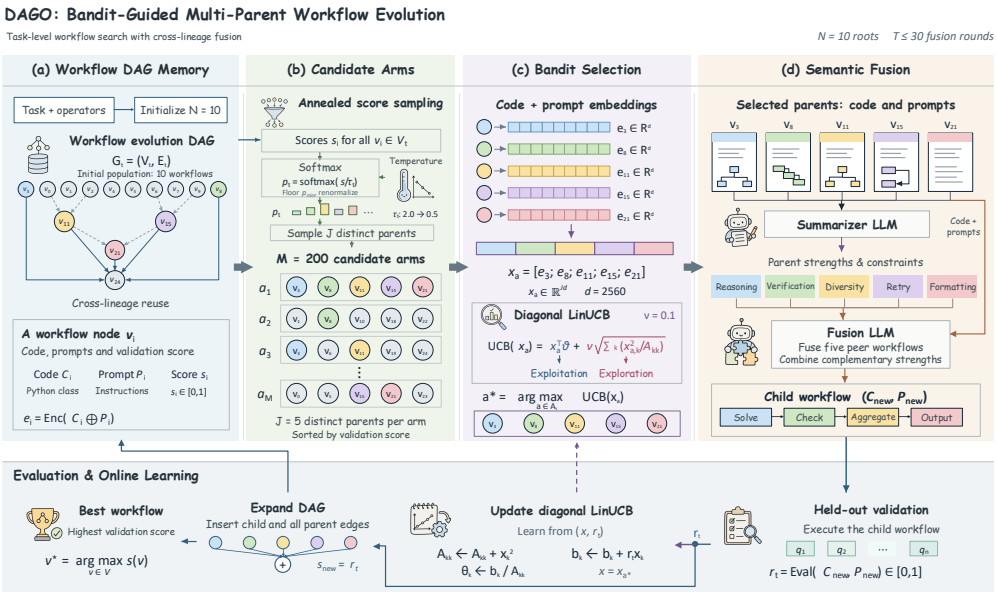


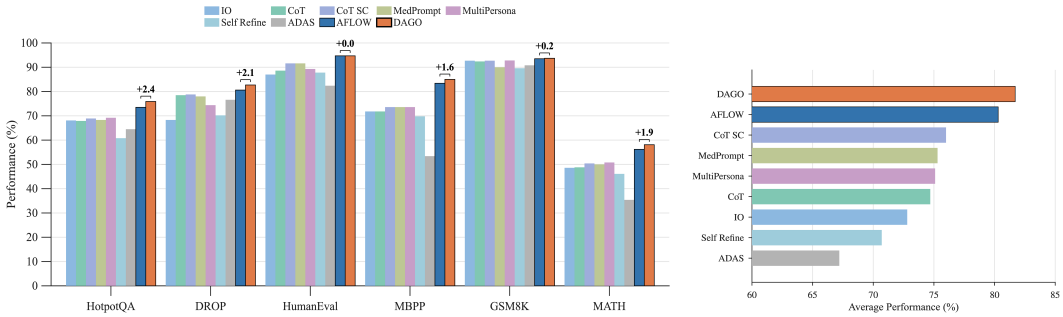
Figure 1: **Overview of the DAGO framework. (a)** The DAG stores workflows, validation scores, and parent lineage. **(b)** Annealed score sampling proposes parent combinations. **(c)** Diagonal LinUCB selects a combination from code-and-prompt embeddings. **(d)** Summary-guided fusion generates a child. Validation feedback updates LinUCB and the DAG; the highest-scoring workflow is returned.

(Directed Acyclic Graph Optimization), a contextual-bandit-guided framework that makes this selection problem explicit. An *arm* is a selectable action in a bandit problem; here, it corresponds to a candidate combination of parent workflows to fuse. The child's validation score serves as the reward, rather than an aggregate of the parents' existing scores. This distinction directs learning toward the quality of the generated workflow, rather than merely favoring parents that already perform well individually.

To learn from limited feedback, DAGO uses a *shared linear reward model* across arms instead of learning an independent reward estimate for every combination. Each workflow is represented by pretrained embeddings of its code and prompts, and an arm's features concatenate its parent embeddings in descending order of validation score. At each iteration, annealed score-based sampling proposes a manageable set of candidate arms without enumerating the full combination space. A diagonal LinUCB policy (Li et al., 2010; Chu et al., 2011) then selects an arm by combining predicted child-workflow quality with an uncertainty-driven exploration bonus. The shared representation allows feedback from evaluated arms to inform the ranking of newly proposed arms, while the exploration bonus encourages evaluating alternatives beyond those with the highest current predictions. The diagonal approximation keeps arm scoring and online updates lightweight. Thus, parent scores guide *candidate proposal*, whereas observed fusion rewards guide *arm selection*, allocating evaluation opportunities through an explicit exploration–exploitation mechanism.

DAGO embeds this decision process in a workflow-generation loop. A shared directed acyclic graph (DAG) stores discovered workflows and their multi-parent generation lineage, providing an expanding pool of parents for subsequent arm proposals. This DAG records the search history, not the execution graph within an individual workflow. After the bandit selects an arm, an LLM summarizes the parents' strengths and constraints and generates a child through summary-guided fusion, starting from the strongest parent and applying targeted modifications informed by the selected workflows. The child's validation score updates the bandit, and the child is added to the DAG. At the end of search, DAGO returns the workflow with the highest validation score for reuse on the target task; arm selection occurs during optimization rather than separately for each test query.

We evaluate DAGO on six benchmarks spanning mathematical reasoning (GSM8K and MATH), code generation (HumanEval and MBPP), and question answering (HotpotQA and DROP). DAGO achieves the highest macro-average score among the evaluated baselines. Under the same validation-evaluation budget cap as AFlow, it improves the macro-average score from 80.3 to 81.7 while



(a) Performance across benchmarks.

(b) Macro-average ranking.

Figure 2: **Overall performance across benchmarks and average ranking.** Selected methods from Table 1, with test scores averaged over three executions. (a) Solve rate / pass@1 / F1 for math / code / QA; brackets show DAGO's gains over AFlow in points. (b) Six-task macro averages.

reducing aggregate search expenditure by 11.2% across the six benchmarks. Arm-selection ablations show that the default LinUCB policy achieves higher mean scores than both random selection and an exploration-free variant on all three tested benchmarks, supporting the value of feedback-guided selection and the exploration bonus. Our contributions are threefold:

1. **A bandit formulation of workflow fusion decisions.** We model each parent workflow combination as an arm whose reward is the generated child's validation score, making feedback-driven parent selection central to workflow search under limited evaluation budgets.
2. **A shared, uncertainty-aware arm-selection mechanism.** We combine pretrained code-and-prompt representations, score-based candidate proposal, and diagonal LinUCB to reuse fusion feedback across arms and guide exploration without exhaustive combination evaluation.
3. **An integrated framework with empirical evaluation.** We integrate arm selection with DAG-based workflow memory and summary-guided fusion, and evaluate performance, search expenditure, and the effects of learned selection and exploration across reasoning, coding, and question-answering tasks.

2 Related Work

Automated Workflow Optimization and Recombination. Automated workflow optimization spans prompt refinement and structural search. DSPy (Khattab et al., 2024), TextGrad (Yuksekgonul et al., 2025), and MIPRO (Opsahl-Ong et al., 2024) improve prompts in modular LLM programs, while GPTSwarm (Zhuge et al., 2024) and G-Designer (Zhang et al., 2025c) optimize multi-agent graphs. ADAS (Hu et al., 2025) searches executable programs using an archive of discoveries, and AFlow (Zhang et al., 2025d) refines code workflows through an MCTS variant with score-based mixed-probability selection. JudgeFlow (Ma et al., 2026) targets problematic blocks using execution diagnostics. ScoreFlow (Wang et al., 2025) and FlowReasoner (Gao et al., 2025) train workflow generators through preference optimization and reinforcement learning, respectively. Multi-parent recombination is also explored: EvoFlow (Zhang et al., 2025a) retrieves query-relevant parents for LLM-based crossover, whereas MermaidFlow (Zheng et al., 2025) applies constraint-preserving crossover to workflow graphs. Building on these approaches, DAGO focuses on *learning which parent combinations to fuse* from past fusion outcomes, allocating a limited evaluation budget to promising generation decisions rather than relying solely on parent scores or query relevance.

Bandit-Guided Selection under Limited Evaluation Budgets. Bandits provide a framework for allocating evaluations through exploration–exploitation. OPTS (Ashizawa et al., 2025) applies Thompson sampling to prompt-design strategy selection, while TRIPLE (Shi et al., 2024) formalizes prompt selection as fixed-budget best-arm identification. MBA-RAG (Tang et al., 2025) selects retrieval strategies, and PILOT (Panda et al., 2025) uses contextual bandits for budget-constrained model routing. Linear contextual bandits (Li et al., 2010; Chu et al., 2011) leverage observed features for reward prediction and uncertainty-aware selection. DAGO applies this principle to workflow fusion: each *arm* corresponds to a parent workflow combination, and its reward is the generated

child's validation score, not the parents' existing scores. Unlike query-time routing among existing models, arm selection guides the generation of new workflows during task-level search. As newly discovered workflows introduce additional arms, the selector must learn from sparse feedback in an expanding search space. A shared linear reward model over pretrained code-and-prompt embeddings allows feedback from evaluated arms to inform newly proposed arms, while diagonal LinUCB balances predicted child quality with uncertainty-driven exploration. This addresses the growing combination space without separately learning each arm's reward or exhaustively evaluating all combinations.

3 PRELIMINARIES

Workflows and validation feedback. A workflow $w = (\mathcal{C}, \mathcal{P})$ comprises executable orchestration code and its internal prompts. Following the code-based representation of AFlow (Zhang et al., 2025d), $\mathcal{C}$ implements an asynchronous `__call__` method that maps a task instance to an output through LLM calls, tools, and control flow. Let $\mathrm{Eval}_q(w) \in [0, 1]$ be its validation score averaged over q executions of the fixed validation set. A stored score $s(v)$ is an observed evaluation, not the unknown expected performance of that workflow. Validation feedback is available during search; test data are not used for generation, arm selection, or stopping.

Workflow archive and fusion arms. The archive is a DAG $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ whose nodes store code, prompts, scores, and cached embeddings. An edge records that a parent was supplied when generating a child; it does not certify that a specific component was inherited. The DAG records generation lineage, not workflow execution. Each *arm* $a = (v_{(1)}, \ldots, v_{(J)})$ specifies J distinct parents, canonically ordered by decreasing score, with node identifiers breaking ties. Let $\bar{\mathcal{A}}_t$ contain all such arms available before round t, and let $\mathcal{A}_t \subseteq \bar{\mathcal{A}}_t$ be the proposed candidates. We use a fixed arm size with $N \geq J$ initial workflows. Canonical ordering removes arbitrary permutations of the same parent set, so the full arm space contains $\binom{|\mathcal{V}_{t-1}|}{J}$ choices before round t.

Budgeted, partially observed decisions. Pulling arm a_t generates one child w_t through LLM summarization and fusion, followed by validation with reward $r_t = \mathrm{Eval}_{q_t}(w_t)$. Conditional on the information $\mathcal{H}_t$ available before selection, define

$$\mu_t(a) = \mathbb{E}[r_t(a) \mid \mathcal{H}_t], \qquad Nq_{\text{init}} + \sum_{t=1}^{K} q_t \leq B_{\max}, \tag{1}$$

where $r_t(a)$ is the potential reward of generating and evaluating a child from a, and K is the number of completed fusion rounds. Only the selected arm's reward is observed. We do not assume that μ_t is linear in arm features or stationary across rounds. The objective is to discover a high-scoring workflow within the evaluation budget and return $\arg\max_{v \in \mathcal{V}_K} s(v)$; per-round reward prediction is a search surrogate, not an equivalent best-workflow guarantee.

4 METHODOLOGY

DAGO (Directed Acyclic Graph Optimization) learns which parent workflow combinations to evaluate through a shared, LinUCB-style selection rule. Each round separates inexpensive candidate proposal and scoring from expensive child generation and evaluation. The DAG supplies reusable parents, the fusion operator generates a child, and bandit feedback guides which arm receives the next evaluation. Lineage is retained for provenance; graph topology itself is not an additional input to the arm scorer in this implementation.

Initialization and candidate proposal. Task-specific strategy discovery initializes N workflows, each scored with q_{init} validation executions and embedded once before insertion into $\mathcal{G}_0$. Initial scores guide parent proposal but are not treated as rewards of unexecuted fusion arms. Before round t, let $s_i = s(v_i)$ for $v_i \in \mathcal{V}_{t-1}$. DAGO computes

$$p_t(v_i) = \frac{\exp(s_i/\tau_t)}{\sum_j \exp(s_j/\tau_t)}, \qquad \widetilde{p}_t(v_i) = \frac{\max\{p_t(v_i), p_{\min}\}}{\sum_j \max\{p_t(v_j), p_{\min}\}}, \tag{2}$$

with $\tau_t = \tau_{\text{init}} + (\tau_{\text{final}} - \tau_{\text{init}})t/T$. At round t, set $M_t = \min\{M, \binom{|\mathcal{V}_{t-1}|}{J}\}$. DAGO repeatedly draws J parents *without replacement within each arm*, canonically orders them, and rejects duplicates until M_t unique arms are collected. The floor preserves positive parent probabilities after renormalization, but does not guarantee that a useful arm appears within a finite budget. Candidate generation uses existing scores; it requires no new workflow evaluation.

Shared arm representation. A fixed encoder maps each node to $\mathbf{e}_i = \text{Enc}(\mathcal{C}_i \oplus \mathcal{P}_i) \in \mathbb{R}^d$, where $\oplus$ concatenates text. DAGO caches these embeddings and represents an arm as $\mathbf{x}(a) = [\mathbf{e}_{(1)}; \ldots; \mathbf{e}_{(J)}] \in \mathbb{R}^p$, with $p = Jd$. Consistent ordering distinguishes the strongest parent from auxiliary parents. A shared linear score transfers feedback across arms instead of maintaining an independent estimate for every combination. Each observed child updates weights subsequently used to score other arms, including newly formed arms whose parents share feature coordinates with earlier candidates. Only one of the M_t proposed arms yields a fusion-reward observation per round, motivating reuse of a fixed representation rather than training a new encoder from these sparse observations. This is a lightweight *reward surrogate*: its additive form does not explicitly model interactions between different parents, and pretrained embeddings do not make LLM fusion linearly realizable.

Diagonal LinUCB-style arm selection. Inspired by linear contextual bandits (Li et al., 2010; Chu et al., 2011), DAGO maintains diagonal statistics $\mathbf{D}_{t-1} = \text{diag}(d_{t-1,1}, \ldots, d_{t-1,p})$ and $\mathbf{b}_{t-1}$, initialized as $\mathbf{D}_0 = \lambda\mathbf{I}$ and $\mathbf{b}_0 = \mathbf{0}$. It computes $\widehat{\boldsymbol{\theta}}_{t-1} = \mathbf{D}_{t-1}^{-1}\mathbf{b}_{t-1}$ and selects

$$a_t \in \arg\max_{a \in \mathcal{A}_t} U_t(a),$$

$$U_t(a) = \underbrace{\mathbf{x}(a)^\top \widehat{\boldsymbol{\theta}}_{t-1}}_{\widehat{\mu}_t(a)} + \nu \underbrace{\sqrt{\sum_{k=1}^{p} x_k(a)^2 / d_{t-1,k}}}_{h_t(a)}. \tag{3}$$

The first term ranks arms using past fusion rewards; the second favors features with less accumulated coordinate-wise coverage. Here h_t is an exploration proxy, not a calibrated confidence interval for nonlinear fusion rewards. Moreover, replacing the full Gram matrix by its diagonal changes both the fitted coefficients and the exploration term; the update is not ordinary multivariate ridge regression. An arm with individually weaker parents can outrank another candidate when its learned score or exploration bonus is larger. However, the selector cannot recover a promising arm omitted by proposal. Scoring M_t arms costs $O(M_t p)$ and storing the bandit state costs $O(p)$.

Fusion, evaluation, and feedback update. For each selected parent, a summarization LLM produces $z_j = \text{Summarize}(\mathcal{C}_j, \mathcal{P}_j, s_j)$ describing strengths, weaknesses, and interface constraints. The fusion LLM receives *both* the parents and their summaries:

$$w_t = \text{Fuse}\big(\{(\mathcal{C}_j, \mathcal{P}_j, s_j, z_j)\}_{v_j \in a_t}\big), \qquad r_t = \text{Eval}_{q_t}(w_t). \tag{4}$$

The prompt starts from the strongest parent and permits a small number of justified, targeted modifications informed by the selected workflows. The reward is the child's absolute validation score, *not* improvement over a parent. With $\mathbf{x}_t = \mathbf{x}(a_t)$, DAGO updates

$$d_{t,k} = d_{t-1,k} + x_{t,k}^2, \qquad b_{t,k} = b_{t-1,k} + r_t x_{t,k}, \quad k = 1, \ldots, p. \tag{5}$$

The evaluated child is embedded and inserted with incoming edges from its selected parents. These sufficient statistics implement the feedback update; no periodic refitting or forgetting step is required by this procedure.

Budget and stopping. Algorithm 1 returns the highest-scoring stored workflow. The reported configuration evaluates $N = 10$ roots once and each child three times, allowing at most $T = 30$ fusion rounds within $B_{\max} = 100$ validation-set evaluations. After at least $T_{\min} = 20$ fusion rounds, early stopping checks for a lack of relative improvement in the most recent $C_{\text{patience}} = 5$ round scores; this recent-score criterion is separate from retaining the highest-scoring workflow. Repeated executions are averaged into *one* reward per selected arm, not three independent fusion decisions. The budget therefore measures validation executions, not proposed arms or generated children alone; LLM generation and summarization costs are accounted for separately. Appendix E specifies sampling, stopping, and implementation details.

Algorithm 1 DAGO: feedback-guided fusion under an evaluation budget

Require: $N \geq J$, $M \geq 1$; $T, B_{\max}$; repeats q_{init}, q; λ, ν; stopping parameters
1: Generate, evaluate, and embed N roots; initialize $\mathcal{G}_0$
2: $\mathbf{D} \leftarrow \lambda \mathbf{I}_{Jd}$, $\mathbf{b} \leftarrow \mathbf{0}$; $B \leftarrow N q_{\text{init}}$
3: Set incumbent $v^* \leftarrow \arg\max_{v \in \mathcal{V}_0} s(v)$; initialize empty reward history
4: **for** $t = 1$ to T **do**
5: **if** $B + q > B_{\max}$ **then break**
6: Set $M_t \leftarrow \min\{M, \binom{|\mathcal{V}_{t-1}|}{J}\}$; sample M_t unique arms using Eq. (2)
7: Select a_t using Eq. (3); set $\mathbf{x}_t \leftarrow \mathbf{x}(a_t)$
8: Summarize selected parents; fuse them into child w_t
9: $r_t \leftarrow \text{Eval}_q(w_t)$; $B \leftarrow B + q$
10: Update $(\mathbf{D}, \mathbf{b})$ by Eq. (5)
11: Embed and insert child with parent edges; update v^* and record r_t
12: **if** $t \geq T_{\min}$ and recent scores satisfy Eq. (34) **then break**
13: **end for**
14: **return** v^*

Decision analysis without linear realizability. We analyze the selector's actual scores rather than transferring a full-matrix linear-bandit guarantee to the LLM pipeline. Let $a_t^\star \in \arg\max_{a \in \mathcal{A}_t} \mu_t(a)$ and define the residual span $\varepsilon_t = \max_{a \in \mathcal{A}_t}(\mu_t(a) - \widehat{\mu}_t(a)) - \min_{a \in \mathcal{A}_t}(\mu_t(a) - \widehat{\mu}_t(a))$. For arbitrary, possibly nonlinear μ_t, the maximizing rule implies

$$\mu_t(a_t^\star) - \mu_t(a_t) \leq \varepsilon_t + \nu\big(h_t(a_t) - h_t(a_t^\star)\big) \leq \varepsilon_t + \nu h_t(a_t). \tag{6}$$

Appendix A proves this inequality, separates candidate-proposal loss, and bounds accumulated diagonal exploration bonuses. The residual span is unchanged by a common offset in all predictions, emphasizing relative discrimination rather than perfect reward calibration. The result identifies prediction differences and proposal coverage as conditions for effective selection; it does not assume they improve or establish sublinear regret for DAGO. The practical value of feedback and exploration is assessed by the random-selection and exploration-free ablations, rather than inferred from linear realizability.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate DAGO on six benchmarks: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) for mathematical reasoning, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) for code generation, and HotpotQA (Yang et al., 2018) and DROP (Dua et al., 2019) for question answering. We follow AFlow's dataset splits and evaluation protocols (Zhang et al., 2025d), restricting MATH to level-5 problems. Validation data are used for search and workflow selection; final results are reported on the disjoint test splits.

Baselines. We compare with prompting and answer-refinement methods: IO, CoT (Wei et al., 2022), CoT-SC (Wang et al., 2023), MedPrompt (Nori et al., 2023), MultiPersona (Wang et al., 2024), Self-Refine (Madaan et al., 2023), and GoT (Besta et al., 2024). We also include answer ensembling with Blender (Jiang et al., 2023), dynamic agent collaboration with DyLAN (Liu et al., 2023), and automated workflow optimization with GPTSwarm (Zhuge et al., 2024), ADAS (Hu et al., 2025), MaAS (Zhang et al., 2025b), and AFlow (Zhang et al., 2025d). AFlow is our main reference for single-parent workflow search. Blender aggregates candidate answers at inference time, whereas DAGO fuses executable workflows during search.

Metrics. We report solve rate (%) on GSM8K and MATH, pass@1 (%) on HumanEval and MBPP (Chen et al., 2021), and F1 (%) on HotpotQA and DROP. The overall score is the unweighted mean across benchmarks. Unless otherwise stated, each validation-selected workflow is fixed and executed three times on the same held-out test split; we report the mean test score. These repetitions are test executions, not independent optimization runs.

Implementation Details. DAGO follows AFlow's fixed model configuration: Claude 3.5 Sonnet (Anthropic, 2024) for workflow generation, summarization, and fusion, and GPT-4o-mini-

Table 1: **Main results on six benchmarks.** Means over three test executions (%), grouped by task category. Avg. is the six-task macro average; best scores are **bold**.

	QA		Code		Math		
Method	HotpotQA	DROP	HumanEval	MBPP	GSM8K	MATH	Avg.
IO	68.1	68.3	87.0	71.8	92.7	48.6	72.8
CoT Wei et al. (2022)	67.9	78.5	88.6	71.8	92.4	48.8	74.7
CoT-SC Wang et al. (2023)	68.9	78.8	91.6	73.6	92.7	50.4	76.0
MedPrompt Nori et al. (2023)	68.3	78.0	91.6	73.6	90.0	50.0	75.3
MultiPersona Wang et al. (2024)	69.2	74.4	89.3	73.6	92.8	50.8	75.1
Self-Refine Madaan et al. (2023)	60.8	70.2	87.8	69.8	89.6	46.1	70.7
GoT Besta et al. (2024)	72.5	71.7	86.3	68.6	86.7	39.5	70.9
Blender Jiang et al. (2023)	72.3	80.4	88.7	77.1	88.4	46.9	75.6
DyLAN Liu et al. (2023)	74.1	82.2	90.4	77.3	90.0	48.5	77.1
GPTSwarm Zhuge et al. (2024)	73.2	81.0	89.3	77.4	89.1	47.9	76.3
ADAS Hu et al. (2025)	64.5	76.6	82.4	53.4	90.8	35.4	67.2
MaAS Zhang et al. (2025b)	75.3	**83.1**	91.6	82.2	91.5	52.2	79.3
AFlow Zhang et al. (2025d)	73.5	80.6	**94.7**	83.4	93.5	56.2	80.3
DAGO	**75.9**	82.7	**94.7**	**85.0**	**93.7**	**58.1**	**81.7**

0718 (OpenAI, 2024) for execution. Both methods use identical validation/test splits and a budget of up to 100 validation-set evaluations per run (20×5 for AFlow; $10 + 30 \times 3$ for DAGO). Workflow selection uses validation scores only; repeated test evaluations are performed after selection and are not used to revise the workflow. Qwen3-Embedding (Zhang et al., 2025e) produces 2560-dimensional workflow embeddings. We set N=10, T=30, M=200, J=5, and ν=0.1, annealing the sampling temperature from 2.0 to 0.5. Full hyperparameters, early-stopping rules, and dataset details appear in Appendices B and C.

5.2 Overall Performance and Analysis

Overall Performance. As shown in Table 1 and Figure 2, DAGO achieves the highest macro-average score of 81.7, compared with 80.3 for AFlow and 79.3 for MaAS. It ranks first on HotpotQA, MBPP, GSM8K, and MATH, and ties AFlow on HumanEval. The gains over AFlow are largest on HotpotQA (+2.4 points) and DROP (+2.1), followed by MATH (+1.9) and MBPP (+1.6); the gain on GSM8K is smaller (+0.2). MaAS remains strongest on DROP, scoring 83.1 versus DAGO's 82.7. Compared with MaAS, DAGO's largest gains are on MATH (+5.9), HumanEval (+3.1), and MBPP (+2.8). Thus, DAGO's average advantage comes from improvements across task categories, rather than a gain on a single benchmark.

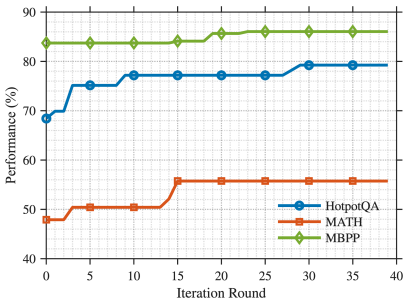


Figure 3: **Validation best-so-far trajectory during optimization.** Running best validation scores on HotpotQA, MATH, and MBPP.

Optimization Trajectory. Figure 3 reports the *validation* best-so-far performance during optimization. Across all three tasks, DAGO exhibits a monotonic improvement trend in the running best solution and typically plateaus by the end of the budget. This pattern indicates that bandit-guided multi-parent fusion steadily discovers and retains higher-quality workflow structures, rather than relying on occasional gains from unguided perturbations.

Zero-Shot Cross-Task Transfer. Figure 4 and Table 2 evaluate whether the best workflows discovered on a source task generalize to a related target task *without* re-optimization. Across all three transfer pairs, DAGO yields consistently lower transfer regret (both absolute and normalized), suggesting that multi-parent fusion tends to distill reusable workflow components that are less tied to source-task idiosyncrasies.

Search Cost and Inference Cost–Performance Trade-offs. Table 3 separates search expenditure from inference cost. DAGO reduces total search cost from $86.45 (AFlow) to $76.78, an 11.2%

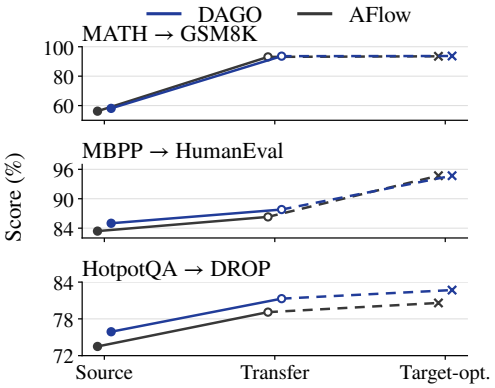


Figure 4: **Cross-task transfer trajectories.** Source / Transfer: source-optimized workflow evaluated on source / target tasks. Target-opt.: target-specific search with the same budget. Transfer uses no re-optimization.

Table 2: **Cross-task transfer.** Means over three test executions: target-optimized Opt_t, zero-shot $\mathrm{Trans}_{s\to t}$, regret $R = \mathrm{Opt}_t - \mathrm{Trans}_{s\to t}$, and normalized regret $\widetilde{R} = R/\mathrm{Opt}_t$. Avg. covers three pairs.

Method	$\mathrm{Opt}_t \uparrow$	$\mathrm{Trans}_{s\to t} \uparrow$	$R_{s\to t} \downarrow$	$\widetilde{R}_{s\to t} \downarrow$
MATH → GSM8K				
AFlow	93.5	93.1	0.4	0.0043
DAGO	**93.7**	**93.6**	**0.1**	**0.0011**
MBPP → HumanEval				
AFlow	**94.7**	86.3	8.4	0.0887
DAGO	**94.7**	**87.8**	**6.9**	**0.0729**
HotpotQA → DROP				
AFlow	80.6	79.1	1.5	0.0186
DAGO	**82.7**	**81.3**	**1.4**	**0.0169**
Avg. (AFlow)	—	—	3.43	0.0372
Avg. (DAGO)	—	—	**2.80**	**0.0303**

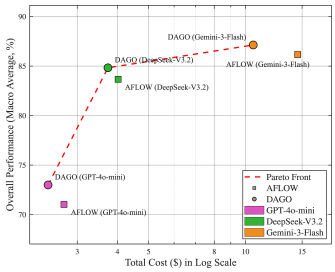
Table 3: **Search cost (USD).** Total covers six benchmarks; lower costs are bold.

Task	AFlow	DAGO
HotpotQA	**13.39**	18.89
DROP	**4.85**	5.64
HumanEval	2.63	**1.92**
MBPP	9.30	**6.93**
GSM8K	20.52	**18.18**
MATH	35.76	**25.22**
Total	86.45	**76.78**

Table 4: **Inference performance and cost.** Means over three test executions. Overall / Cost: three-task average / total USD. Hotpot: HotpotQA. Better results per executor are bold.

Method	MATH	MBPP	Hotpot	Overall	Cost
GPT-4o-mini OpenAI (2024)					
AFlow	56.2	83.4	73.5	71.0	2.7278
DAGO	**58.1**	**85.0**	**75.9**	**73.0**	**2.4344**
DeepSeek-V3.2 Liu et al. (2025)					
AFlow	85.8	87.1	78.1	83.7	4.0197
DAGO	**86.3**	**87.7**	**80.5**	**84.8**	**3.7368**
Gemini-3-Flash Google (2025)					
AFlow	**87.0**	92.1	79.4	86.2	14.5263
DAGO	86.8	**93.0**	**81.6**	**87.1**	**10.5712**

Figure 5: **Inference cost–performance.** Table 4 with log-scale cost; dashed line: Pareto frontier.



saving overall. It is cheaper on four benchmarks, with the largest saving on MATH ($10.54), but more expensive on HotpotQA and DROP, so the reduction is aggregate rather than uniform. For inference, Table 4 and Figure 5 show that DAGO achieves a higher three-task average on MATH, MBPP, and HotpotQA at lower cost for all execution models: costs decrease by 10.8%, 7.0%, and 27.2% with GPT-4o-mini, DeepSeek-V3.2, and Gemini-3-Flash, while average scores increase from 71.0 to 73.0, 83.7 to 84.8, and 86.2 to 87.1, respectively. These inference comparisons exclude search expenditure and cover only the three reported test sets.

5.3 Ablation Study

Effect of Bandit-Guided Arm Selection. Table 6 compares LinUCB with random selection and different exploration strengths. Random selection lowers the score by 2.9 points on GSM8K, 3.1 on HumanEval, and 1.9 on DROP relative to the default ν=0.1. Both greedy selection (ν=0) and stronger exploration (ν=1) also yield lower means on all three tasks. Among the tested settings, moderate exploration gives the best results; neither random nor greedy arm selection matches it.

Effect of Individual Components. Table 7 evaluates four components on GSM8K, HotpotQA, and MBPP. Removing parent summaries or replacing UCB with random selection reduces the three-task average from 84.9 to 82.9 and 83.0, respectively, while single-parent fusion yields 83.3 and incurs the largest drop on HotpotQA (2.1 points), supporting the use of parent summaries, multi-parent fusion, and bandit-guided selection. Full-matrix LinUCB achieves 84.2, below the diagonal variant's 84.9; with high-dimensional arm features and at most 30 fusion rewards per search, estimating cross-feature covariances may be less reliable, while the diagonal approximation also avoids maintaining a full $Jd \times Jd$ covariance matrix.

Figure 6: **Effect of arm size J.** Means over three test executions with varying numbers of parents.

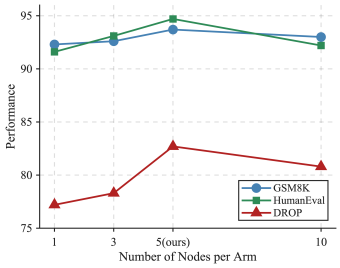


Table 5: **Robustness across execution models.** Means over three test executions for the selected workflows.

Model	Method	MATH	MBPP	HotpotQA
GPT-4o-mini	AFlow	56.2	83.4	73.5
	DAGO	**58.1**	**85.0**	**75.9**
DeepSeek-V3.2	AFlow	85.8	87.1	78.1
	DAGO	**86.3**	**87.7**	**80.5**
Gemini-3-Flash	AFlow	**87.0**	92.1	79.4
	DAGO	86.8	**93.0**	**81.6**

Table 6: **Arm selection.** Means over three test executions; ν controls UCB exploration.

Method	GSM8K	HumanEval	DROP
AFlow	93.5	**94.7**	80.6
DAGO (ν=0.1)	**93.7**	**94.7**	**82.7**
DAGO (Random)	90.8	91.6	80.8
DAGO (ν=0)	92.0	93.9	81.4
DAGO (ν=1)	91.7	92.4	82.4

Table 7: **Component ablations.** Means over three test executions; Avg. is the three-task macro average. Best scores are bold.

Setting	GSM8K	HotpotQA	MBPP	Avg.
AFlow	93.5	73.5	83.4	83.5
DAGO (full)	**93.7**	**75.9**	**85.0**	**84.9**
w/o summarization	92.0	74.2	82.6	82.9
w/o UCB (Random)	90.8	74.5	83.7	83.0
Full-matrix LinUCB	93.2	75.1	84.2	84.2
Single-parent only	92.7	73.8	83.5	83.3

Effect of Arm Size. Figure 6 studies the impact of arm size, i.e., the number of parent workflows contributing to each fusion step. When J=1, DAGO degenerates to single-parent evolution and performs worst, suggesting limited structural diversity. Performance improves as J increases and peaks around J=5, while a larger arm (J=10) leads to degradation, consistent with a trade-off: increasing J can enrich complementary signals, but overly many parents may introduce incompatible patterns and dilute the fusion objective.

Robustness to Execution Model Choice. Table 5 indicates that DAGO generally maintains its advantage across execution models, improving on two out of three tasks even when the executor is already strong. This supports the interpretation that DAGO's gains primarily stem from discovering better workflow structures (e.g., decomposition, verification, and coordination patterns), rather than relying on idiosyncrasies of a specific base model.

6 Conclusion

We presented DAGO, a framework for LLM-based workflow optimization that learns which parent workflows to fuse from the validation feedback of their offspring. By treating parent combinations as bandit arms, DAGO uses a shared reward predictor and a diagonal UCB-style policy to guide selection, while a DAG archive and summary-guided fusion support the reuse and refinement of discovered workflows. Across six benchmarks spanning mathematical reasoning, code generation, and question answering, DAGO achieves the highest macro-average score among the evaluated methods. Under the same validation-evaluation budget cap as AFlow, it improves the macro-average score from 80.3 to 81.7 while reducing aggregate search expenditure by 11.2%. Ablations further support the value of feedback-guided selection and exploration, alongside the benefits of multi-parent fusion. Together, these results highlight feedback-guided parent selection as a key component of effective workflow fusion: prior search outcomes inform not only which workflows are available for reuse, but also which combinations to explore next.

AI Use Statement

In this work, we used generative AI tools to improve the clarity, grammar, and presentation of the manuscript. We also used these tools to assist with implementing, debugging, and refining parts of the

research code. All AI-assisted text was reviewed and revised by the authors, and all AI-assisted code was inspected and tested by the authors. The authors take full responsibility for the final manuscript, implementation, and experimental results.

Ethics Statement

This work presents a methodological study of automated agentic workflow optimization using existing language models and standard benchmarks for mathematical reasoning, code generation, and question answering. The study does not involve human participants, private or sensitive personal data, or deployment in high-stakes settings. Our experiments evaluate workflows within established benchmark protocols. We are not aware of any specific ethical concerns arising from the experiments conducted in this study.

Reproducibility Statement

We provide anonymized DAGO source code in the supplementary material, including the optimizer, prompt templates, task operators, and evaluation harness, together with setup and usage instructions. Section 4 and Algorithm 1 describe the DAGO optimization procedure, including candidate-arm proposal, bandit-guided parent selection, and workflow fusion. Appendix A provides the theoretical analysis and its assumptions. Section 5 specifies the models, benchmarks, baselines, evaluation budgets, and reporting protocol. Appendices B and C detail the hyperparameters, stopping criteria, dataset splits, and evaluation metrics. Appendix D presents the LLM prompt templates, and Appendix E provides extended pseudocode for sampling, diagonal LinUCB updates, embedding construction, and the full optimization loop. Appendices F and G document discovered workflows, their code and prompts, and detailed experimental results. These descriptions specify the search and evaluation settings used in our experiments and provide concrete workflow examples for reimplementation.

References

Yasin Abbasi-yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/e1d5be1c7f2f456670de3d53c7b54f4a-Paper.pdf.

Anthropic. Introducing claude 3.5 sonnet. URL=https://www.anthropic.com/news/claude-3-5-sonnet, 2024.

Rin Ashizawa, Yoichi Hirose, Nozomu Yoshinari, Kento Uchida, and Shinichi Shirakawa. Bandit-based prompt design strategy selection improves prompt optimizers. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20799–20817, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1070. URL https://aclanthology.org/2025.findings-acl.1070/.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michał Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, Mar 2024. doi: 10.1609/aaai.v38i16.29720. URL https://ojs.aaai.org/index.php/AAAI/article/view/29720.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In Geoffrey Gordon, David Dunson, and Miroslav Dudík (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 208–214, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/chu11a.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*, 2025.

Avishek Ghosh, Sayak Ray Chowdhury, and Aditya Gopalan. Misspecified linear bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Google. Gemini 3 flash: frontier intelligence built for speed. URL=https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792/.

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. URL https://openreview.net/forum?id=sY5N0zY5Od.

Tor Lattimore, Csaba Szepesvari, and Gellert Weisz. Learning with good feature representations in bandits and in RL with a generative model. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5662–5670. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/lattimore20a.html.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pp. 661–670, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772758. URL https://doi.org/10.1145/1772690.1772758.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization, 2023.

Zihan Ma, Zhikai Zhao, Chuanbo Hua, Federico Berto, and Jinkyoo Park. Judgeflow: Agentic workflow optimization via block judge. *arXiv preprint arXiv:2601.07477*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, et al. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *arXiv preprint arXiv:2311.16452*, 2023.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. URL=https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

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 Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.525. URL https://aclanthology.org/2024.emnlp-main.525/.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive LLM routing under budget constraints. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23934–23949, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1301. URL https://aclanthology.org/2025.findings-emnlp.1301/.

Chengshuai Shi, Kun Yang, Zihan Chen, Jundong Li, Jing Yang, and Cong Shen. Efficient prompt optimization through the lens of best arm identification. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=FLNnlfBGMo.

Xiaqiang Tang, Qiang Gao, Jian Li, Nan Du, Qi Li, and Sihong Xie. MBA-RAG: a bandit approach for adaptive retrieval-augmented generation through question complexity. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3248–3254, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.218/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. Scoreflow: Mastering llm agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*, 2025.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 257–279, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.15. URL https://aclanthology.org/2024.naacl-long.15/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

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 Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

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.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*, 2025a.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=imcyVlzpXh.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025c. URL https://openreview.net/forum?id=Jov79pGXc6.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025d. URL https://openreview.net/forum?id=z5uVAKwmjf.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025e.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming. *arXiv preprint arXiv:2505.22967*, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

Appendix

Appendix A Theoretical Perspective on Bandit-Guided Multi-Parent Fusion 15

Appendix B Hyperparameter Settings 19

Appendix C Dataset Details 20

Appendix D LLM Prompt Templates 21

Appendix E Extended Algorithm Details 24

Appendix F Comprehensive Analysis of Discovered Workflows 28

Appendix G Full Experimental Results 39

A Decision Analysis without Linear Realizability

This appendix characterizes the implemented selection rule without assuming that LLM-driven fusion rewards are linear in pretrained embeddings. Its results separate what follows algebraically from the rule, what requires accurate predictions, and what remains an empirical question. They are not a new general-purpose bandit algorithm or a regret guarantee for the complete workflow-generation process.

A.1 History, potential rewards, and comparison scope

Immediately before round t, the archive is $\mathcal{G}_{t-1}$. Let $\bar{\mathcal{A}}_t$ contain all canonically ordered arms of J distinct parents from that archive. The proposal mechanism repeatedly samples arms and rejects duplicates until the candidate set $\mathcal{A}_t$ contains $M_t = \min\{M, |\bar{\mathcal{A}}_t|\}$ distinct arms, where M is the candidate cap. Let $\mathcal{H}_t$ include previous decisions and rewards, the archive, cached features, fixed model configurations, the current bandit state, and the realized candidates, before new fusion randomness is drawn. For every feasible arm a, define the potential reward

$$r_t(a) = \mathrm{Eval}_{q_t}\big(\mathrm{Fuse}(a; \xi_t^{\mathrm{gen}}); \xi_t^{\mathrm{eval}}\big), \qquad \mu_t(a) = \mathbb{E}[r_t(a) \mid \mathcal{H}_t] \in [0,1]. \tag{7}$$

Here fusion includes summarization and the full parent inputs; the random variables include any remaining generation and execution variability. The observed reward is $r_t = r_t(a_t)$. Averaging several executions of the *same generated child* reduces evaluation variability only to the extent permitted by their dependence; it does not average over independently generated children. No independence between executions, sub-Gaussian parameter, stationary reward law, or realizable linear function is needed for the pathwise inequalities below.

Write $m_t(a) = \widehat{\mu}_t(a)$, $h_t(a) \geq 0$, and $U_t(a) = m_t(a) + \nu h_t(a)$ for the actual diagonal scores of Eq. (3). The notation m_t avoids conflating a fitted score with the conditional mean μ_t. The fitted score need not lie in $[0,1]$. Throughout, a_t maximizes U_t over the proposed candidates, with a fixed tie-breaking rule.

A one-step comparator, not an oracle search trajectory. The full-arm comparator uses the archive *actually reached by DAGO* at that round. An optimizer making different earlier selections would generally create a different archive and face different arms. Consequently, the analysis does not compare DAGO with the best possible sequence of workflow-generation decisions or with the globally optimal program. Expanding the archive changes the available arms; it does not, by itself, imply that the conditional reward of a fixed arm changes. The notation μ_t nevertheless allows changes in the reward law.

A.2 Separating candidate proposal from arm selection

Let

$$\bar{a}_t^\star \in \arg\max_{a \in \bar{\mathcal{A}}_t} \mu_t(a), \qquad a_t^\star \in \arg\max_{a \in \mathcal{A}_t} \mu_t(a). \tag{8}$$

Define the proposal gap $g_t = \mu_t(\bar{a}_t^\star) - \mu_t(a_t^\star) \geq 0$ and selection gap $\Delta_t = \mu_t(a_t^\star) - \mu_t(a_t) \geq 0$. For any realized run of K completed rounds,

$$R_K^{\mathrm{archive}} := \sum_{t=1}^{K} \big(\mu_t(\bar{a}_t^\star) - \mu_t(a_t)\big) = \underbrace{\sum_{t=1}^{K} g_t}_{\text{proposal loss}} + \underbrace{\sum_{t=1}^{K} \Delta_t}_{R_K^{\mathrm{sel}}}. \tag{9}$$

This is an exact identity for arbitrary reward functions. No selector restricted to $\mathcal{A}_t$ can eliminate loss caused by missing a useful arm during proposal.

Proposition A.1 (Coverage with rejection sampling to distinct arms)**.** *Condition on the history before proposal and on the round's full-arm reward function. Let $F_t = |\bar{\mathcal{A}}_t|$, $M_t = \min\{M, F_t\}$, and let Q_t be a fixed distribution with positive mass on every feasible arm. Draw independent* raw proposals *from Q_t, rejecting repetitions until M_t distinct arms have been accepted. For $\gamma \geq 0$, define*

$$\mathcal{S}_{t,\gamma} = \{a : \mu_t(a) \geq \max_{b \in \bar{\mathcal{A}}_t} \mu_t(b) - \gamma\}, \qquad q_{t,\gamma} = Q_t(\mathcal{S}_{t,\gamma}). \tag{10}$$

If proposal randomization does not change the reward distribution of a fixed arm, then

$$\Pr(g_t > \gamma) \leq (1 - q_{t,\gamma})^{M_t} \leq e^{-M_t q_{t,\gamma}}, \qquad \mathbb{E}[g_t] \leq \gamma + e^{-M_t q_{t,\gamma}}. \tag{11}$$

The first relation is an inequality, not the equality for M_t independent draws with replacement.

Proof. Every feasible arm has positive probability, so collecting $M_t \le F_t$ distinct arms terminates almost surely. Couple this procedure with the same infinite sequence of independent raw proposals and let $L_t \ge M_t$ be the number consumed before termination. Every arm appearing in the first M_t raw proposals belongs to the final accepted set: it is either accepted on first appearance or was accepted earlier. Therefore, if the final candidate set misses $\mathcal{S}_{t,\gamma}$, the first M_t raw proposals must also miss it. The latter event has probability $(1 - q_{t,\gamma})^{M_t}$, proving the first inequality. The inequality $1 - u \le e^{-u}$ gives the exponential bound. Finally, $0 \le g_t \le 1$ implies $\mathbb{E}[g_t] \le \gamma + \Pr(g_t > \gamma)$. □

Accepted arms are dependent. Let $\mathcal{U}_{m-1}$ be the first $m-1$ accepted arms. For $a \notin \mathcal{U}_{m-1}$, rejection sampling gives

$$\Pr(A_m = a \mid \mathcal{U}_{m-1}) = \frac{Q_t(a)}{1 - Q_t(\mathcal{U}_{m-1})}, \qquad Q_t(\mathcal{U}) = \sum_{b \in \mathcal{U}} Q_t(b). \tag{12}$$

Previously accepted arms have probability zero. Thus accepted candidates are neither independent nor, in general, uniformly sampled. For example, if Q_t is uniform and $s = |\mathcal{S}_{t,\gamma}|$, the exact miss probability is $\binom{F_t - s}{M_t} / \binom{F_t}{M_t}$, with value zero when $F_t - s < M_t$. In particular, collecting the entire finite arm space gives $g_t = 0$.

The reward-invariance condition concerns the specified fusion interface, which receives only the selected parents and their summaries, not the other candidates; it also requires proposal draws not to alter the conditional fusion-randomness law. The coupling comparison holds for the same Q_t and the same target of M_t candidates, not for the same number of raw sampling attempts or the same wall-clock proposal cost. Rejection sampling may require substantially more than M_t raw proposals, although these attempts consume no validation evaluations. Moreover, positive support alone does not make $q_{t,\gamma}$ large or ensure a useful numerical bound. No realizable linear reward law is needed for this coverage result.

A.3 A DECISION BOUND FOR AN ARBITRARY NONLINEAR REWARD FUNCTION

Define the actual prediction residual and its span over the current candidates:

$$e_t(a) = \mu_t(a) - m_t(a), \qquad \varepsilon_t = \max_{a \in \mathcal{A}_t} e_t(a) - \min_{a \in \mathcal{A}_t} e_t(a). \tag{13}$$

This is a property of the fitted score on the current candidates, not an assumption that the true reward has a small linear approximation error. The span measures errors in *predicted reward differences*: it is unchanged by adding the same constant to all predictions. If $|e_t(a)| \le \epsilon_t$ happens to hold for all candidates, then $\varepsilon_t \le 2\epsilon_t$, but no such calibration assumption is imposed.

Proposition A.2 (Selection error of the implemented score)**.** *For any nonempty candidate collection, any real-valued predictor m_t, any nonnegative exploration proxy h_t, any $\nu \ge 0$, and any reward function $\mu_t : \mathcal{A}_t \to [0,1]$, the maximizer of $m_t + \nu h_t$ satisfies*

$$0 \le \Delta_t \le e_t(a_t^\star) - e_t(a_t) + \nu\big(h_t(a_t) - h_t(a_t^\star)\big) \le \varepsilon_t + \nu h_t(a_t). \tag{14}$$

Proof. Since $U_t(a_t) \ge U_t(a_t^\star)$, $m_t(a_t^\star) - m_t(a_t) \le \nu(h_t(a_t) - h_t(a_t^\star))$. Adding $e_t(a_t^\star) - e_t(a_t)$ to both sides gives the first upper bound. The residual difference is at most its span and $h_t(a_t^\star) \ge 0$, giving the second bound. □

What the bound explains. A shared predictor is useful when its reward differences help discriminate promising arms, even if its absolute predictions are biased. Exploration can deliberately choose an arm with lower predicted reward to collect feedback from less-covered features. Its immediate decision cost is visible in Eq. (14); any subsequent benefit must appear through better future selection or predictions. The proposition does *not* prove that exploration reduces future residual spans or that embeddings encode all relevant parent interactions.

A ranking-margin interpretation. For any suboptimal candidate a, it cannot be selected whenever

$$\mu_t(a_t^\star) - \mu_t(a) > e_t(a_t^\star) - e_t(a) + \nu\big(h_t(a) - h_t(a_t^\star)\big), \tag{15}$$

since then $U_t(a_t^\star) > U_t(a)$. This relates a selection decision to its prediction-difference error and exploration preference without attributing linear structure to the fusion operator.

A.4 ACCUMULATED DIAGONAL EXPLORATION BONUSES

The following calculation concerns the *implemented* statistic, rather than a full-matrix confidence ellipsoid. It requires a finite feature-norm bound and unchanged additive diagonal updates; it does not require any reward model. For a finite run one may use $L = \max_{1\le t\le K} \|\mathbf{x}_t\|_2$ in the statement; claiming an asymptotic rate would require a uniform bound as the horizon grows. No extra feature normalization is introduced into DAGO by this analysis.

Proposition A.3 (A pathwise bound on diagonal bonuses)**.** *Suppose $d_{0,k} = \lambda > 0$, $d_{t,k} = d_{t-1,k} + x_{t,k}^2$, and $\|\mathbf{x}_t\|_2 \le L$. Let $h_t(a_t)^2 = \sum_k x_{t,k}^2/d_{t-1,k}$ and $c_{\lambda,L} = 1 + L^2/\lambda$. Then*

$$\sum_{t=1}^{K} h_t(a_t) \le \sqrt{K c_{\lambda,L} \sum_{k=1}^{p} \log\left(1 + \frac{\sum_{t=1}^{K} x_{t,k}^2}{\lambda}\right)} \le \sqrt{K c_{\lambda,L} p \log\left(1 + \frac{KL^2}{p\lambda}\right)}. \tag{16}$$

Proof. Set $u_{t,k} = x_{t,k}^2/d_{t-1,k}$. Because $d_{t-1,k} \ge \lambda$, $0 \le u_{t,k} \le L^2/\lambda$. The elementary inequality $\log(1+u) \ge u/(1+u)$ gives $u_{t,k} \le c_{\lambda,L} \log(1+u_{t,k})$. Therefore,

$$\sum_{t=1}^{K} h_t(a_t)^2 = \sum_{t=1}^{K}\sum_{k=1}^{p} u_{t,k} \tag{17}$$

$$\le c_{\lambda,L} \sum_{k=1}^{p}\sum_{t=1}^{K} \log \frac{d_{t,k}}{d_{t-1,k}} = c_{\lambda,L} \sum_{k=1}^{p} \log \frac{d_{K,k}}{\lambda}. \tag{18}$$

Cauchy–Schwarz yields the first inequality in Eq. (16). Concavity of the logarithm and $\sum_{t,k} x_{t,k}^2 \le KL^2$ yield the second. □

Corollary A.4 (Error-dependent archive-relative decision bound)**.** *Under the update and feature conditions of Proposition A.3,*

$$R_K^{\text{archive}} \le \sum_{t=1}^{K} g_t + \sum_{t=1}^{K} \varepsilon_t + \nu \sqrt{K c_{\lambda,L} p \log\left(1 + \frac{KL^2}{p\lambda}\right)}. \tag{19}$$

The bound also holds at an early-stopped horizon because each ingredient is pathwise.

Proof. Combine Eq. (9), Proposition A.2, and Proposition A.3. □

This is an *error-dependent diagnostic*, not an assertion of sublinear regret. Only the accumulated exploration proxy is bounded directly by the chosen update. Neither $\sum_t g_t$ nor $\sum_t \varepsilon_t$ is shown to be sublinear, and either can grow linearly. For fixed p, λ, L, ν, sublinear growth of both terms would be sufficient for vanishing average archive-relative gaps; the reported workflow experiments do not establish those premises. Moreover, with $p = Jd = 12{,}800$ and at most 30 fusion decisions in the reported configuration, the numerical bound can be vacuous relative to the elementary bound $R_K^{\text{archive}} \le K$. It should not be presented as a quantitative explanation of the observed gains.

A.5 WHY STANDARD LINEAR-BANDIT GUARANTEES DO NOT TRANSFER AUTOMATICALLY

Classical linear-bandit confidence arguments rely on a realizable reward model, appropriately controlled noise and feature norms, a full Gram matrix, and a compatible confidence radius (Abbasi-yadkori et al., 2011). Misspecification is not innocuous: existing work demonstrates failure modes and feature-dimension-dependent approximation effects (Ghosh et al., 2017; Lattimore et al., 2020). DAGO instead uses a fixed exploration coefficient and a diagonal approximation to both estimation and exploration. Simply adding an unspecified linear-approximation term to a classical rate does not establish a theorem for this rule.

Both the prediction and the exploration metric change. For the same observed data, define

$$\mathbf{G}_t = \lambda \mathbf{I} + \sum_{s=1}^{t} \mathbf{x}_s \mathbf{x}_s^\top, \qquad \mathbf{D}_t = \operatorname{diag}(\mathbf{G}_t), \qquad \mathbf{b}_t = \sum_{s=1}^{t} r_s \mathbf{x}_s. \tag{20}$$

The full ridge solution is $\widehat{\boldsymbol{\theta}}_t^{\text{full}} = \mathbf{G}_t^{-1}\mathbf{b}_t$. DAGO uses $\widehat{\boldsymbol{\theta}}_t^{\text{diag}} = \mathbf{D}_t^{-1}\mathbf{b}_t$. Thus, analyzing only the difference between $\sqrt{\mathbf{x}^\top \mathbf{G}_t^{-1}\mathbf{x}}$ and $\sqrt{\mathbf{x}^\top \mathbf{D}_t^{-1}\mathbf{x}}$ omits a simultaneous change in the fitted mean.

An explicit correlated-feature example. Take n noiseless observations with feature $(1,1)^\top$, reward 1, and $\lambda > 0$. These observations are compatible with the exactly linear parameter $(1/2, 1/2)^\top$. Then

$$\mathbf{G}_n = \begin{pmatrix} \lambda + n & n \\ n & \lambda + n \end{pmatrix}, \qquad \mathbf{D}_n = (\lambda + n)\mathbf{I}, \qquad \mathbf{b}_n = (n, n)^\top. \tag{21}$$

At the observed feature, the diagonal fitted value is $2n/(\lambda + n) \to 2$, whereas the full ridge fitted value is $2n/(\lambda + 2n) \to 1$. The diagonal estimate is therefore not automatically consistent even on these realizable data. For the unobserved contrast $\mathbf{x}_- = (1, -1)^\top$,

$$\mathbf{x}_-^\top \mathbf{G}_n^{-1} \mathbf{x}_- = 2/\lambda, \qquad \mathbf{x}_-^\top \mathbf{D}_n^{-1} \mathbf{x}_- = 2/(\lambda + n) \to 0. \tag{22}$$

Repeated observations of the common direction make the diagonal exploration proxy shrink even in a contrast direction whose full-matrix uncertainty has not decreased. Conversely, along $(1,1)^\top$ the diagonal proxy exceeds the full one. There is no universal conservative ordering between the two metrics.

A conditional comparison when correlations are small. For completeness, let $\mathbf{C}_t = \mathbf{D}_t^{-1/2} \mathbf{G}_t \mathbf{D}_t^{-1/2} - \mathbf{I}$. If the measurable condition $\|\mathbf{C}_t\|_2 \le \rho < 1$ holds, spectral bounds give

$$\frac{\mathbf{x}^\top \mathbf{D}_t^{-1} \mathbf{x}}{1 + \rho} \le \mathbf{x}^\top \mathbf{G}_t^{-1} \mathbf{x} \le \frac{\mathbf{x}^\top \mathbf{D}_t^{-1} \mathbf{x}}{1 - \rho}. \tag{23}$$

Also,

$$|\mathbf{x}^\top (\widehat{\boldsymbol{\theta}}_t^{\mathrm{diag}} - \widehat{\boldsymbol{\theta}}_t^{\mathrm{full}})| \le \frac{\rho}{1 - \rho} \, \|\mathbf{D}_t^{-1/2} \mathbf{x}\|_2 \|\mathbf{D}_t^{-1/2} \mathbf{b}_t\|_2. \tag{24}$$

Indeed, the eigenvalues of $\mathbf{I} + \mathbf{C}_t$ lie in $[1 - \rho, 1 + \rho]$, and $\|\mathbf{I} - (\mathbf{I} + \mathbf{C}_t)^{-1}\|_2 \le \rho/(1 - \rho)$. These yield Eqs. (23)–(24) by a change of coordinates and Cauchy–Schwarz. No small-correlation condition is assumed for the reported experiments. The calculation identifies what would need to be checked rather than restoring a missing guarantee.

A.6 Representation limits and the final-workflow objective

Additivity does not represent general parent interactions. Partitioning $\widehat{\boldsymbol{\theta}}$ into J blocks yields

$$m_t(a) = \sum_{j=1}^{J} \mathbf{e}_{(j)}^\top \widehat{\boldsymbol{\theta}}_{t-1,j}. \tag{25}$$

Thus the prediction is additive across parent roles and has no explicit products between parent embeddings. For a two-role example with scalar parent features $u, v \in \{0, 1\}$, the reward $\mu(u, v) = uv$ cannot be represented by any additive model $c + \alpha u + \beta v$: the first three input pairs force $c = \alpha = \beta = 0$, contradicting the reward at $(1, 1)$. This illustrates why the method is described as a useful selection surrogate, not a learned model of arbitrary structural complementarity. Interaction features or nonlinear surrogates are possible extensions, but are not part of the reported algorithm.

Expected fusion reward and best observed child are different objectives. Consider a fixed archive with arm a always producing reward 0.6 and arm b producing 1 with probability 0.5 and 0 otherwise, independently across pulls. The higher mean is $\mu(a) = 0.6 > 0.5 = \mu(b)$, so always selecting a has zero cumulative regret to the mean-optimal arm. Yet after $K \ge 2$ pulls, the expected best reward from b is $1 - 2^{-K} > 0.6$. Consequently, small selection regret does not imply optimal best-child discovery, even in a stationary two-arm example. Returning the largest stored validation score also does not guarantee the largest true test performance, particularly with adaptive validation reuse or unequal evaluation repetition counts.

A.7 Evidence and falsifiable diagnostics

The existing random-arm and $\nu = 0$ ablations assess whether feedback-guided selection and the exploration term help end-to-end search under the reported protocol. They do not estimate g_t or ε_t, prove linearity, or calibrate h_t as a confidence interval. The following diagnostics specify additional tests; *no new numerical results from these tests are claimed here*.

Prequential prediction records. Save every candidate arm, its features, prediction, exploration proxy, proposal seed, selected arm, and eventual reward *before* updating the model. Evaluate predictions against subsequently observed rewards, not against the same rewards after refitting. Selected-arm residuals are useful but do not reveal errors on unselected arms or certify the residual span.

Frozen-archive, multi-arm probes. At selected checkpoints, freeze the archive, candidate collection, model state, prompts, and execution settings. Allocate a separately reported diagnostic budget to multiple randomly sampled candidate arms and independently generated children per arm. Repeated executions of each child separate execution variation from generation variation. Compare predicted differences, empirical reward differences, and the observed quality of arms selected by random, greedy, and exploration-aware rules. Finite probes can estimate local ranking behavior, but cannot identify the full-archive optimum or uniformly bound unobserved arms.

Matched-budget end-to-end controls. Use the same initialization protocol, proposal mechanism, execution models, validation splits, and evaluation accounting for random, greedy, diagonal, and full-matrix selectors. Report repeated-run variation as well as mean outcomes. A full-matrix ridge control can be implemented through the low-rank identity in Appendix E.6; it need not allocate a dense $p \times p$ matrix. Any additional diagnostic evaluations must be separated from the search budget used for headline comparisons.

B HYPERPARAMETER SETTINGS

This appendix summarizes the hyperparameter configuration of DAGO used in all experiments. Unless otherwise noted, we use the default settings in Table 8 across benchmarks. For reproducibility, we organize hyperparameters by: (i) search budget and early stopping, (ii) candidate-arm proposal, (iii) LinUCB guidance, (iv) representations and embeddings, (v) LLM roles and decoding, and (vi) evaluation.

Table 8: DAGO hyperparameters (default unless otherwise specified).

Category	Parameter	Value	Description / usage
Budget	Initial population size N	10	Number of diverse workflows generated at initialization (inserted into the DAG before bandit-guided fusion).
	Max iterations T	30	Maximum number of fusion iterations in Algorithm 1.
	Stopping window C_{patience}	5	Check for a lack of relative improvement over the most recent five fusion-round validation scores.
	Minimum iterations $T_{\min}$	20	Enable early stopping only after at least $T_{\min}$ iterations.
Arm proposal	Candidate-arm cap M	200	Propose $M_t = \min\{M, \binom{\lvert\mathcal{V}_{t-1}\rvert}{J}\}$ distinct parent tuples in round t.
	Parents per arm J	5	Number of parent workflows fused to synthesize one child.
	Temperature $\tau_{\text{init}} \to \tau_{\text{final}}$	$2.0 \to 0.5$	Linear temperature annealing for score-based softmax proposal (Appendix E.2).
	Minimum probability $p_{\min}$	0.01	Probability floor to prevent proposal collapse and ensure coverage over all nodes.
	Within-arm duplicates	No	Parents are sampled without replacement; J is fixed during each search, with $N \geq J$.
LinUCB	Exploration coefficient ν	0.1	Exploration weight in Eq. (3); ν is kept constant by default.
	Regularization λ	0.2	Ridge regularization in the bandit covariance (Eq. (3)).
	Covariance approximation	Diagonal	Maintain only $\mathrm{diag}(\mathbf{A})$ to compute uncertainty in $O(Jd)$ (Appendix E.3).
	Reward r	$[0, 1]$	Child validation score, normalized to $[0, 1]$, used as the bandit reward.

Table 8: DAGO hyperparameters (continued).

Category	Parameter	Value	Description / usage
Embeddings	Embedding dimension d	2560	Dimension of the semantic embedding vector $\mathbf{e}_i$ for each workflow node.
	Embedding model	Qwen3	Pretrained code embedding model encoding `code` $\oplus$ `inner_prompts`.
	Arm feature $\mathbf{x}$	Concat	$\mathbf{x} = [\mathbf{e}_{m_1}; \ldots; \mathbf{e}_{m_J}] \in \mathbb{R}^{Jd}$ (Appendix E.3).
LLMs	Optimization LLM	Claude-3.5-Sonnet	Used for strategy discovery, initial workflow generation, parent summarization, and fusion.
	Execution LLM	GPT-4o-mini	Used to execute and evaluate candidate workflows during search and final reporting.
	Generation temperature	0.0	Decoding temperature for workflow generation/fusion (summarization typically uses a lower temperature for stability).
Evaluation	Validation samples	Dataset-specific	Fixed full validation splits: GSM8K 264; MATH 119; HumanEval 33; MBPP 86; HotpotQA 200; DROP 200 (Table 9).
	Test samples	Full set	Full test split for final reporting.
	Max concurrency	50	Maximum number of concurrent evaluations (API/hardware dependent).
	Evaluations per node	1 / 3	Root nodes are evaluated once; fused offspring are evaluated 3 times and averaged to reduce variance.

C Dataset Details

This section provides additional dataset information beyond Section 5, with a focus on the *online reward estimation* protocol used during search. To ensure a fair, apples-to-apples comparison, **we strictly follow the AFlow evaluation protocol for (i) dataset partitioning, (ii) answer extraction / code execution, and (iii) metric computation**. Concretely, for each benchmark we use the same "divided test set" setup as AFlow: a *fixed* held-out validation subset is used exclusively to provide online reward feedback during optimization, while the final numbers are reported on a disjoint test subset. (Zhang et al., 2025d)

Table 9: Benchmark datasets and evaluation protocol used in our experiments (aligned with AFlow). **Val Size** is the fixed validation subset used for reward estimation during search (held fixed across iterations), and **Test Size** is the disjoint evaluation subset used for reporting final results.

Dataset	Domain	Val Size	Test Size	Metric	Notes
GSM8K	Math Reasoning	264	1,055	Accuracy	Grade-school word problems
MATH	Math Reasoning	119	486	Accuracy	Level 5 (hardest) problems only
HumanEval	Code Generation	33	131	Pass@1	Python function synthesis
MBPP	Code Generation	86	341	Pass@1	Mostly Basic Python Problems
HotpotQA	Question Answering	200	800	F1	Multi-hop reasoning required
DROP	Question Answering	200	800	F1	Discrete reasoning over paragraphs

Evaluation Metrics (standard metrics; computed with the AFlow harness). We adopt the *canonical* metrics used by these benchmarks, and compute them using the same evaluation pipeline as AFlow. (Zhang et al., 2025d) For mathematical reasoning (GSM8K, MATH), we report exact-match accuracy (a.k.a. solve rate) after answer extraction. GSM8K is evaluated by extracting a final numeric answer from the model output and matching it to the ground truth (with a small numeric tolerance to handle formatting). (Cobbe et al., 2021) For MATH, the benchmark is designed to be scored by exact match after normalization of the final answer, and we follow the standard practice of extracting the final boxed answer and checking equivalence (including symbolic checking for algebraic expressions when needed). (Hendrycks et al., 2021)

For code generation (HumanEval, MBPP), we report Pass@1, the standard unit-test-based functional correctness metric: a solution is counted as correct if the first generated program passes all provided

tests in a sandboxed environment. This is the established evaluation protocol for HumanEval and widely used for MBPP-style program synthesis benchmarks. (Chen et al., 2021; Austin et al., 2021) We apply code sanitization to extract the target function and run tests with timeouts, matching the AFlow setup. (Zhang et al., 2025d)

For question answering (HotpotQA, DROP), we compute token-level F1 after standard text normalization (lowercasing, article removal, punctuation stripping, and whitespace normalization). HotpotQA and DROP are commonly reported with both EM and F1 in their official evaluations; following AFlow, we use F1 as the primary metric in our main tables. (Yang et al., 2018; Dua et al., 2019; Zhang et al., 2025d)

Sampling Strategy. Following AFlow, validation instances are randomly sampled *once* and then held fixed across all iterations, so that online rewards are comparable over time and across methods. (Zhang et al., 2025d) For MATH, we restrict to Level 5 (the hardest subset) to obtain a challenging optimization target. Every validation execution evaluates the entire fixed validation subset in Table 9: 264, 119, 33, 86, 200, and 200 instances for GSM8K, MATH, HumanEval, MBPP, HotpotQA, and DROP, respectively. No additional per-round validation subsampling is used. Final evaluation is performed on the full held-out test subset in Table 9.

D LLM Prompt Templates

This section summarizes the core LLM prompt templates used in DAGO. We employ role-specialized prompts for: (i) *strategy discovery*, (ii) *initial population generation*, (iii) *parent advantage summarization*, and (iv) *multi-parent fusion*. To make the optimization process robust, all generation prompts enforce a strict XML response schema that is machine-parseable into `workflow_code` and `prompt` fields (Appendix E).

Prompt formatting (reproducibility). For readability, we present prompts using a consistent boxed style with syntax highlighting.

D.1 Shared Constraints and Output Schema

All optimizer-facing prompts append a shared constraint block that (a) defines the required workflow interface, (b) documents the operator return conventions, (c) specifies evaluator interface requirements (output format and parsing conventions), and (d) enforces the XML output schema.

Shared Constraints and Interface Specification

```
=== CONSTRAINTS ===
Workflow: `class Workflow(__init__(llm), async __call__(problem, **kwargs)->str)`
CRITICAL - All operators return dict, MUST extract value:
  Custom/CustomCodeGenerate/ScEnsemble -> `result['response']` (str)
  Programmer -> `result['output']` (str)
  Test -> `result['solution']` (str), `result['result']` (bool)
WRONG: `result.split()`, `for x in result`  # result is dict, not str!
RIGHT: `result['response'].split()`, `for x in result['response'].split('\n')`
Inner prompts: Plain strings, NO `{problem}` placeholders - input is auto-appended.

=== EVALUATOR INTERFACE ===
{evaluator_interface}  # output format spec: answer extraction pattern, code
↪  conventions

=== OUTPUT FORMAT ===
<workflow_code>
import asyncio
import inner_prompt as inner_prompts
from dago_operators import Custom  # add other operators as needed

class Workflow:
    def __init__(self, execution_llm):
        self.llm = execution_llm
        self.prompts = inner_prompts
        self.custom = Custom(execution_llm)

    async def __call__(self, problem: str, **kwargs) -> str:
        # For code tasks (HumanEval/MBPP): MUST extract and use entry_point!
        # entry_point = kwargs.get('entry_point', '')
```

```
        # instruction must be f-string: instruction=f"...function named
        ↪  '{entry_point}'..."
        result = await self.custom(input=problem,
        ↪  instruction=self.prompts.SOLVE_PROMPT)
        return result['response']  # MUST extract from dict!
</workflow_code>

<prompt>
SOLVE_PROMPT = """Analyze and solve step by step. Output ONLY the final answer."""
</prompt>

Prompt names in <prompt> must match those used in workflow code.
```

D.2 Strategy Discovery Prompt

The strategy discovery prompt generates diverse workflow strategies for a given task type:

Strategy Discovery Template

```
Generate **{num_strategies}** structurally distinct LLM workflow strategies
for the **{task_display_name}** benchmark.

**Context**: You are designing LLM-based workflows to solve problems from
the {task_display_name} dataset. Each strategy should describe how to
orchestrate LLM calls to solve these problems effectively.

**Requirements**:
1. **Topological Diversity**: Strategies must differ in **control flow**
   (One-Shot vs. Linear Chain vs. Recursive/Loop).
2. **Mechanism Fit**: Address specific failure modes of this benchmark
   (e.g., Code -> Test-Driven; Math -> Step-Verification).
3. **Simplicity First**: Default to the simplest effective flow.

**Output JSON**:
[{
  "name": "Strategy Name",
  "control_flow_type": "One-Shot|Linear|Recursive|Tree",
  "core_approach": "Brief execution steps",
  "task_adaptation": "Specific failure mode addressed",
  "complexity_profile": "low|medium|high"
}]
```

D.3 Initial Population Generation Prompt

Given a sampled strategy, DAGO prompts the optimizer LLM to generate an executable workflow and its internal prompt set:

Initial Workflow Generation Template

```
Design a workflow for **{task_type}** using strategy: **{strategy_choice}**.

Workflow Design:
- Target 1-5 LLM calls (max 8 if truly needed)
- Each call must serve a distinct, necessary purpose
- Simpler workflows often perform better than complex ones

Inner Prompt Design (EQUALLY IMPORTANT):
- Inner prompts determine output quality - design them carefully
- The ENDING of each prompt is critical - it controls output format
- Follow the output format in EVALUATOR INTERFACE

{shared_constraints}  # includes operator docs + evaluator interface + XML schema
```

D.4 Parent Advantage Summarization Prompt

Before fusion, DAGO analyzes each parent workflow to extract strengths and weaknesses:

Parent Advantage Summarization Template

```
Analyze this workflow for **{task_type}** (score: {score}).

Workflow Code:
```python
{workflow_code}
```
Inner Prompts:
```python
{inner_prompts_code}
```

Three-Dimensional Analysis:
1. **STRUCTURE**: How many LLM calls? What does each do?
2. **INNER PROMPTS**: Quote the exact ending instruction of each prompt.
3. **EVALUATOR ALIGNMENT**: Will the output format be correctly parsed?

Output YAML:
structure:
  llm_calls: N
  flow: "linear|conditional|iterative"
  each_call_purpose: ["call 1 does X", "call 2 does Y"]
prompt_analysis:
  final_prompt_ending: "Quote the last 1-2 sentences"
  format_instruction_clear: yes|no
  potential_format_issues: ["issue 1", "issue 2"]
evaluator_alignment:
  output_will_parse_correctly: yes|no|uncertain
  risks: ["specific parsing risk 1"]
strengths:
  - "what works and why"
weaknesses:
  - what: "specific problem"
    fix: "concrete fix suggestion"
    severity: "low|medium|high"
fusion_guidance:
  must_preserve: ["exact behavior to keep"]
  safe_to_modify: ["what can be adjusted"]
  dangerous_to_modify: ["do not touch these"]
```

D.5 Multi-Parent Fusion Prompt

The fusion prompt synthesizes a new workflow by combining strengths from multiple parents:

Multi-Parent Fusion Template

```
Create a workflow for **{task_type}** by improving the BEST parent.

**Parent Workflows** (sorted by score, highest first):
{parent_workflows_info}

=== MANDATORY FUSION PROTOCOL ===

**Step 1: COPY the highest-scoring parent as your BASELINE**
- Your default output should be IDENTICAL to the best parent
- Do not "rewrite" or "improve" unless you have concrete evidence

**Step 2: Review each parent's FULL analysis summary**
- Understand structure (llm_calls, flow), prompt_analysis, and evaluator_alignment
- Pay attention to `weaknesses` and `fusion_guidance`
- Identify at most 2-3 SMALL, TARGETED changes that address documented issues

**Step 3: For EACH proposed change, answer:**
- [ ] What specific failure does this fix? (cite from parent analysis)
- [ ] Could this break something that currently works?
- [ ] Is this change truly necessary, or am I adding complexity for no reason?
- If ANY answer is uncertain -> DO NOT make this change

**Step 4: Inner Prompt Changes (CRITICAL)**
- If changing a prompt ending, quote BOTH the original and your new version
```

```
- Explain why the new wording better matches evaluator behavior
- The last 1-2 sentences of each prompt are the most important

=== CHANGE BUDGET ===
- **Modifications**: Maximum 2-3 targeted changes
- **LLM calls**: Keep similar to best parent (<=8 total); adding calls requires
↪  justification

Types of allowed changes:
1. Fix format instruction - make prompt ending match evaluator expectations
2. Remove unnecessary complexity - delete redundant LLM calls
3. Fix documented bug - address specific weakness from parent analysis

Types of FORBIDDEN changes:
- Adding new verification steps "just to be safe"
- Rewriting prompts that already work well
- Changing control flow without evidence of improvement
- Adding complexity because it "sounds better"

=== EVALUATOR INTERFACE FOR {task_type} ===
{evaluator_interface}  # output format requirements

{shared_constraints}

**OUTPUT**: Think through your changes first, then provide the XML output.
```

E Implementation Details of Bandit-Guided Fusion

This appendix specifies the selection loop corresponding to Eqs. (2)–(5). It distinguishes workflow evaluations from arm scoring, preserves the diagonal numerical update, and makes the budget and feature dimensions explicit.

E.1 State, dimensions, and reported configuration

Each node stores an immutable identifier, workflow code $\mathcal{C}_i$, internal prompts $\mathcal{P}_i$, an observed validation score s_i, the evaluation repetition count, its fixed embedding $\mathbf{e}_i$, and parent identifiers. The active node pool includes the initial workflows and subsequently evaluated children. The bandit state consists of vectors $\mathbf{d}, \mathbf{b} \in \mathbb{R}^p$ with $p = Jd$; node embeddings and graph edges are stored separately.

Table 10: Reported settings used to specify the revised procedure. The table does not introduce new tuned parameters.

Component	Parameter	Reported value
Initialization	N, q_{init}	10, 1
Fusion budget	T, q, $B_{\max}$	30, 3, 100
Stopping	$T_{\min}$, C_{patience}, δ_{stop}	20, 5, 10^{-4}
Candidate cap / arm size	M, J	200, 5
Sampling	$\tau_{\text{init}} \to \tau_{\text{final}}$, $p_{\min}$	$2.0 \to 0.5$, 0.01
Bandit	λ, ν	0.2, 0.1
Features	d, $p = Jd$	2560, 12800
Representation	Encoder family	Qwen3-Embedding
Generation	Optimization model	Claude 3.5 Sonnet
Execution	Evaluation model	GPT-4o-mini-0718

The default configuration satisfies $N \geq J$ and $\binom{N}{J} = \binom{10}{5} = 252$, so $M_t = 200$ throughout the default search. More generally, the candidate count is capped at the number of available parent combinations, while J and the feature dimension remain fixed within each search. For a deployment with fewer than J initial parents, initialization must be expanded or a fixed padding-and-mask convention must be specified before search; silently changing J while retaining a Jd-dimensional state is invalid. Such an extension is not needed for the reported setting and is not added here.

E.2 CANDIDATE-ARM PROPOSAL

Stable score sampling. For the $n_t = |\mathcal{V}_{t-1}|$ available nodes, compute

$$w_t(v) = \exp\left(\frac{s(v) - s_{\max}}{\tau_t}\right), \quad p_t(v) = \frac{w_t(v)}{\sum_u w_t(u)},$$
$$\widetilde{p}_t(v) = \frac{\max\{p_t(v), p_{\min}\}}{Z_t}, \quad Z_t = \sum_u \max\{p_t(u), p_{\min}\}. \tag{26}$$

Subtracting $s_{\max}$ prevents numerical overflow without changing the distribution. The renormalized probabilities obey $\widetilde{p}_t(v) \geq p_{\min}/Z_t > 0$, not necessarily $\widetilde{p}_t(v) \geq p_{\min}$. Since $1 \leq Z_t \leq 1 + n_t p_{\min}$, the floor is a support safeguard, not a fixed lower bound on every arm's inclusion probability. The temperature is computed from the planned maximum T, even when search stops early.

Without replacement inside an arm. To sample one arm, start with all nodes eligible, draw a parent with probability proportional to its remaining weight $\widetilde{p}_t(v)$, remove it, and repeat until J distinct parents have been drawn. Sort the result by decreasing validation score and then increasing node identifier. For an unordered parent set S of size J, the probability of its canonical arm is

$$Q_t(S) = \sum_{\pi \in \mathrm{Perm}(S)} \prod_{j=1}^{J} \frac{\widetilde{p}_t(v_{\pi_j})}{1 - \sum_{\ell<j} \widetilde{p}_t(v_{\pi_\ell})}. \tag{27}$$

Thus parent draws within an arm are not independent. The canonical ordering does not affect membership but fixes the feature positions used by the selector.

Round-dependent number of distinct candidates. Let $F_t = \binom{n_t}{J}$ be the number of feasible parent combinations before round t. The number of distinct candidates is

$$M_t = \min\{M, F_t\} = \min\left\{M, \binom{|\mathcal{V}_{t-1}|}{J}\right\}. \tag{28}$$

The candidate set starts empty at each round. Repeatedly sample one arm from the fixed distribution Q_t in Eq. (27), canonicalize it, and insert it only if its parent-identifier sequence has not already been accepted in this round. Continue until $|\mathcal{A}_t| = M_t$. Thus M is a *cap*, whereas M_t is the actual number of unique candidate arms, not the number of raw sampling attempts. Duplicates are determined by parent identities, not by whether different nodes contain identical workflow text. The same arm may be proposed again in a later round. Raw proposals are independent conditional on the archive and sampling weights, but accepted arms are dependent with the law in Eq. (12); weights are not recomputed after rejected proposals.

When $F_t < M$, all feasible parent combinations are included, without changing J or allowing repeated parents. For the arm-size ablations with $N = 10$ and $M = 200$, the first-round candidate counts for $J \in \{1, 3, 5, 10\}$ are 10, 120, 200, and 1, respectively. Positive arm probabilities ensure almost-sure termination, although there is no deterministic upper bound on the number of raw proposals.

If L_t is the total number of raw proposals and $\mathcal{U}_m$ the first m accepted arms, the next acceptance has geometric waiting time with success probability $1 - Q_t(\mathcal{U}_m)$. Consequently,

$$\mathbb{E}[L_t] = \sum_{m=0}^{M_t-1} \mathbb{E}\left[\frac{1}{1 - Q_t(\mathcal{U}_m)}\right] \leq \frac{1}{q_{\min,t}} \sum_{m=0}^{M_t-1} \frac{1}{F_t - m}, \quad q_{\min,t} = \min_{a \in \bar{\mathcal{A}}_t} Q_t(a) > 0. \tag{29}$$

This finite expectation may be large when proposal mass is concentrated. Appendix A.2 analyzes coverage for this rejection-sampling procedure rather than treating the accepted candidates as independent.

E.3 EMBEDDINGS AND THE DIAGONAL REWARD SURROGATE

Fixed features and cached encodings. Construct $\mathbf{e}(v) = \mathrm{Enc}(\mathcal{C}(v) \oplus \mathcal{P}(v))$ once for each immutable workflow text and cache it. The encoder is not fine-tuned by the bandit. For arm $a = (v_{(1)}, \ldots, v_{(J)})$, concatenate the ordered embeddings into $\mathbf{x}(a) \in \mathbb{R}^{Jd}$. No interaction features,

Algorithm 2 Candidate proposal with a round-dependent cap

Require: Nodes $\mathcal{V}_{t-1}$ with $n_t \geq J$; candidate cap $M \geq 1$; $J, \tau_t, p_{\min}$
$M_t \leftarrow \min\{M, \binom{n_t}{J}\}$
Compute stable normalized weights $\widetilde{p}_t$; $\mathcal{A}_t \leftarrow \varnothing$
while $|\mathcal{A}_t| < M_t$ **do**
  Draw J distinct nodes using sequential weighted sampling without replacement
  Canonically order the nodes by score and identifier to form arm a
  if $a \notin \mathcal{A}_t$ **then**
    $\mathcal{A}_t \leftarrow \mathcal{A}_t \cup \{a\}$
  end if
end while
return $\mathcal{A}_t$

extra intercept, feature normalization, score clipping, or encoder retraining is introduced by the mathematical revision. The encoder checkpoint, serialization of the prompt set, truncation length, and any preprocessing used by an implementation must be fixed across training and arm scoring and reported with the code.

Exact diagonal computation. Before round t, the sufficient statistics are

$$d_{t-1,k} = \lambda + \sum_{s<t} x_{s,k}^2, \qquad b_{t-1,k} = \sum_{s<t} r_s x_{s,k}, \qquad \widehat{\theta}_{t-1,k} = b_{t-1,k}/d_{t-1,k}. \tag{30}$$

Arm predictions and exploration proxies are

$$m_t(a) = \sum_{k=1}^{p} x_k(a)\widehat{\theta}_{t-1,k}, \qquad h_t(a) = \sqrt{\sum_{k=1}^{p} x_k(a)^2/d_{t-1,k}}. \tag{31}$$

The selected arm maximizes $m_t(a) + \nu h_t(a)$. After its child has been evaluated, add $\mathbf{x}_t \odot \mathbf{x}_t$ to $\mathbf{d}$ and $r_t\mathbf{x}_t$ to $\mathbf{b}$. Each fusion decision contributes one observation even when its child's reward averages multiple validation executions. Initial nodes do not create synthetic fusion observations.

Relationship to least squares. The coefficient vector minimizes the separable quadratic

$$\frac{1}{2}\boldsymbol{\theta}^\top \mathbf{D}_{t-1}\boldsymbol{\theta} - \mathbf{b}_{t-1}^\top \boldsymbol{\theta}, \tag{32}$$

which replaces the full design Gram matrix by its diagonal. In contrast, multivariate ridge regression minimizes $\frac{1}{2}\sum_{s<t}(r_s - \mathbf{x}_s^\top\boldsymbol{\theta})^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$ and retains all cross-feature terms. The distinction matters for both coefficients and confidence interpretation.

Cold start, histories, and nonstationarity. Before the first observed fusion reward, $\mathbf{b}_0 = 0$, so selection is driven by $\nu\|\mathbf{x}(a)\|_2/\sqrt{\lambda}$, with identity-based tie breaking. Without a fixed feature-norm convention, the initial preference may reflect embedding norms. Storing past arms and outcomes is useful for auditing, but recomputing Eq. (30) from the same history yields the same state. It is not a mechanism for adapting to reward drift. Forgetting factors, sliding windows, or full-matrix refits would change the algorithm and require separate evaluation.

E.4 LLM INTERACTION AND EVALUATION INTERFACE

Initialization. A strategy-discovery call supplies task-level design ideas to initial workflow generation. Each generated root is evaluated with the fixed validation harness and inserted with its score and embedding. The test set is excluded from these calls.

Summarization. For every selected parent, provide its code, internal prompts, and score to the summarization model. The structured analysis records call structure, prompt endings, evaluator-interface risks, strengths, weaknesses, and restrictions on safe edits. These are LLM-generated analyses rather than verified causal attributions. They are inputs to fusion, not labels for the bandit and not substitutes for child validation.

Constrained multi-parent fusion. The fusion input contains all selected parent code and prompts together with their scores and summaries. The existing template instructs the model to use the highest-scoring parent as the baseline, consider at most two or three targeted changes, and justify any additional calls or control-flow modifications. Accordingly, "fusion" denotes generation conditioned on multiple parents, often through a local edit of the strongest parent; it does not require literal inheritance from every supplied parent. Parent edges record conditioning provenance. The prompt and output-schema constraints are generation instructions, not guarantees that the emitted program is valid or improves performance.

Bounded validation reward. With metric $\ell \in [0, 1]$ and fixed validation set $\mathcal{D}_{\text{val}}$,

$$\text{Eval}_q(w) = \frac{1}{q} \sum_{j=1}^{q} \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_{\text{val}}} \ell\big(w(x; \xi_j), y\big). \tag{33}$$

Each j is an execution of the same generated program over the same split, with any execution randomness included in ξ_j. The task-specific harness defines answer parsing, failed-instance scores, and timeouts. The search score uses the mean over all declared executions, not the maximum or a selectively filtered subset. Failure handling and retries must be fixed consistently across methods: a failed attempt must not be silently dropped to obtain uncharged additional generation or evaluation opportunities. An entire-child failure can be assigned a bounded failure reward by the declared harness; this convention must be documented rather than inferred from the selection equations.

E.5 Budget accounting and early stopping

What is counted. One budget unit is one execution of the complete validation split for a single workflow. With $q_{\text{init}} = 1$ and $q = 3$, a run with K completed fusion rounds uses $B(K) = N + 3K \leq 100$ declared validation-set executions, absent additional charged attempts. Scoring M_t candidate arms uses cached features and is not M_t workflow evaluations. Summarization and fusion still consume LLM resources, so equal validation budgets do not imply equal dollar cost. Any charged retries must be included in the counter before another evaluation is permitted.

Recent-score stopping and incumbent retention. The stopping decision and the returned workflow use different information. The incumbent is the workflow with the highest stored validation score across all initial and fusion nodes; it is updated whenever a new score is strictly higher, retaining the existing incumbent on ties. Early stopping instead checks whether the most recent $C_{\text{patience}} = 5$ fusion-round validation scores show no relative improvement under the stopping tolerance $\delta_{\text{stop}} = 10^{-4}$. Let

$$\mathbf{r}_t^{\text{recent}} = (r_{t-C_{\text{patience}}+1}, \ldots, r_t)$$

denote this recent-score window. Write $r_0 = r_{t-C_{\text{patience}}+1}$ and $r_+ = \max\{r_{t-C_{\text{patience}}+2}, \ldots, r_t\}$. Define

$$\text{Stagnant}(\mathbf{r}_t^{\text{recent}}; \delta_{\text{stop}}) = \begin{cases} (r_+ - r_0)/r_0 \leq \delta_{\text{stop}}, & r_0 > 0, \\ r_+ = 0, & r_0 = 0. \end{cases}$$

Thus the best score in the final four rounds is compared with the first score of the five-round window. A positive score after a zero baseline counts as improvement. The stopping gate is

$$t \geq T_{\min} = 20 \quad \text{and} \quad \text{Stagnant}(\mathbf{r}_t^{\text{recent}}; \delta_{\text{stop}}) = \text{true}. \tag{34}$$

This is not a counter of consecutive failures to exceed the best initial or historical score. The window contains completed fusion-round rewards, each averaged over the declared validation executions; initialization evaluations are not window entries. The minimum applies to completed fusion rounds, not to individual validation executions. The reported budget permits all 20 rounds: $Nq_{\text{init}} + 20q = 70 \leq 100$. The run ends when the recent-score stopping gate is met or the iteration/evaluation budget is exhausted.

E.6 Computational costs and full-matrix comparison

Diagonal selection cost. The bandit stores $O(p)$ statistics, scores one arm in $O(p)$ time, scores M_t arms in $O(M_t p)$ time, and updates in $O(p)$ time. Candidate features can be constructed and scored in a stream, avoiding an additional $O(M_t p)$ materialized feature matrix. The archive additionally uses

Algorithm 3 End-to-end DAGO with explicit budget and stopping state

Require: $N \geq J$, $M \geq 1$; $T, B_{\max}$; q_{init}, q; λ, ν; sampling settings; $T_{\min}, C_{\text{patience}}, \delta_{\text{stop}}$
Check $Nq_{\text{init}} \leq B_{\max}$; initialize empty DAG and embedding cache
Discover task strategies; generate N roots; evaluate each q_{init} times and insert
$B \leftarrow Nq_{\text{init}}$; $p \leftarrow Jd$; $\mathbf{d} \leftarrow \lambda \mathbf{1}_p$; $\mathbf{b} \leftarrow \mathbf{0}_p$
$v^* \leftarrow \arg\max_{v \in \mathcal{V}_0} s(v)$; initialize an empty recent-reward window
for $t = 1$ to T **do**
  if $B + q > B_{\max}$ **then**
    break
  end if
  Compute τ_t and $\widetilde{p}_t$; propose candidates using Algorithm 2
  $\widehat{\boldsymbol{\theta}} \leftarrow \mathbf{b} \oslash \mathbf{d}$
  Form $\mathbf{x}(a)$ for candidates and select $a_t \in \arg\max_a [m_t(a) + \nu h_t(a)]$
  Set $\mathbf{x}_t \leftarrow \mathbf{x}(a_t)$; summarize all selected parents
  Generate child w_t using parent code, prompts, scores, and summaries
  $r_t \leftarrow \text{Eval}_q(w_t)$; $B \leftarrow B + q$; record the decision and outcome
  $\mathbf{d} \leftarrow \mathbf{d} + \mathbf{x}_t \odot \mathbf{x}_t$; $\mathbf{b} \leftarrow \mathbf{b} + r_t \mathbf{x}_t$
  Cache the child's embedding; insert it and all parent edges into the DAG
  if $r_t > s(v^*)$ **then**
    $v^* \leftarrow v_{\text{new}}$
  end if
  Append r_t to the recent-reward window; retain only its last C_{patience} entries
  if $t \geq T_{\min}$ and $\text{Stagnant}(\mathbf{r}_t^{\text{recent}}; \delta_{\text{stop}})$ **then**
    break
  end if
end for
return v^* and the archive

$O(nd)$ embedding storage and $O(KJ)$ parent links, apart from workflow text and logs. Computing parent proposal weights costs $O(n)$. If one raw weighted draw costs $C_{\text{draw}}(n, J)$, proposal costs $O(n + L_t[C_{\text{draw}}(n, J) + J \log J])$ for the realized L_t attempts, using hashing for duplicate rejection. Storing the accepted parent identifiers uses $O(M_t J)$ space. Only the M_t accepted arms need bandit scoring; L_t need not equal M_t. LLM calls, execution, and initial embedding computation are separate from the bandit arithmetic.

Dense full covariance is not the only exact implementation. At $p = 12{,}800$, a dense full Gram matrix has $p^2 = 163{,}840{,}000$ entries. However, with $K \leq 30$ observations, let $\mathbf{X} \in \mathbb{R}^{K \times p}$ contain their features. The Woodbury identity gives

$$(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X})^{-1} = \lambda^{-1} \mathbf{I} - \lambda^{-1} \mathbf{X}^\top (\lambda \mathbf{I}_K + \mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{X}. \tag{35}$$

Thus exact full-ridge predictions and uncertainty can be obtained using an $O(Kp + K^2)$ representation instead of storing p^2 entries. For example,

$$h_{\text{full}}(\mathbf{x})^2 = \lambda^{-1} \left[\|\mathbf{x}\|_2^2 - (\mathbf{X}\mathbf{x})^\top (\lambda \mathbf{I}_K + \mathbf{X}\mathbf{X}^\top)^{-1} (\mathbf{X}\mathbf{x}) \right]. \tag{36}$$

After the small-system factorization, evaluating this expression for a new arm costs $O(Kp + K^2)$ using straightforward matrix-vector products and triangular solves, compared with $O(p)$ for the diagonal proxy. The diagonal choice is therefore a speed-and-modeling trade-off, not evidence that exact full-ridge computation is impossible. An observed difference between diagonal and full-matrix ablations does not by itself establish a statistical explanation for that difference.

Reproducibility boundaries. Candidate de-duplication uses the round-dependent count M_t, and early stopping applies to the most recent five round scores after the $t \geq 20$ gate. Exact reproduction also requires model revisions, decoding settings, validation splits, encoder preprocessing, and failure/retry accounting. These settings should accompany the released implementation and run logs. Additional nonlinear surrogates, interaction features, forgetting mechanisms, or diagnostic probes are not silently included in the reported configuration.

F Comprehensive Analysis of Discovered Workflows

This section presents a detailed case study of the optimal workflows discovered by DAGO across all six benchmark datasets, with comparative analysis against AFlow where applicable. We analyze

workflow structures, multi-parent fusion lineages, emergent design patterns, and provide insights into why DAGO's DAG-based optimization discovers more effective solutions. Unless otherwise specified, scores in this appendix are *validation scores used during optimization*, not the final test results in Table 1. Each initial workflow is evaluated once on the fixed validation split ($q_{\text{init}} = 1$); each fusion workflow is evaluated three times on that same split, and its mean score is stored as the reward ($q = 3$). Separately, the final selected workflow is executed three times on the held-out test split for reporting.

F.1 Cross-Dataset Optimization Statistics

Table 11 summarizes the optimization trajectories across all benchmarks, revealing several important patterns about DAGO's search dynamics.

Table 11: DAGO optimization statistics per benchmark. Initial scores use one execution of the fixed validation split; fusion scores average three executions. Relative improvement is measured against the best initial score. Fusion-reward standard deviations describe variation across evaluated fusion nodes, not variation across independent searches or test executions.

Metric	GSM8K	MATH	HumanEval	MBPP	HotpotQA	DROP
Total Iterations	28	27	25	25	30	30
Early Stopped	Yes	Yes	Yes	Yes	No	No
Total DAG Nodes	38	37	35	35	40	40
Total DAG Edges	140	135	125	125	150	150
Best Initial Score	95.8%	50.4%	87.9%	83.7%	77.2%	82.9%
Best Fusion Score	95.8%	55.7%	88.9%	86.0%	79.3%	84.0%
Relative Improvement	+0.0%	+10.5%	+1.1%	+2.7%	+2.7%	+1.3%
Best Fusion Node ID	11	15	12	23	29	25
Iteration Found	11	15	12	23	29	25
Parents of Best	5	5	5	5	5	5
Mean Fusion Reward	93.9%	49.6%	86.0%	78.4%	70.8%	77.5%
Fusion Reward Std Dev	1.0%	2.7%	2.3%	16.3%	12.0%	6.0%

Key Observations from Optimization Dynamics. Several important patterns emerge from Table 11:

(1) Task-Dependent Stopping Behavior. GSM8K, MATH, HumanEval, and MBPP met the recent-score stopping criterion after 25–28 fusion rounds, whereas HotpotQA and DROP reached the 30-round limit. These are stopping outcomes under the reported search protocol, rather than evidence that a local optimum has been reached. The stopping criterion examines recent round scores separately from the archive-wide best score.

(2) Consistent Multi-Parent Fusion. All best fusion workflows are produced by 5-parent fusion operations (J=5), demonstrating that DAGO's multi-parent mechanism is essential for combining complementary strengths from multiple workflows. This contrasts sharply with AFlow's single-parent mutation, which cannot systematically recombine modules from multiple lineages.

(3) Fusion Timing Patterns. Best workflows were discovered at diverse iteration points (11–29), indicating that premature convergence is not a concern. The bandit-guided exploration successfully balances exploitation of promising regions with exploration of novel combinations.

F.2 Mathematical Reasoning: MATH Dataset

Best MATH Workflow (Node 15, Validation Score: 55.7%). The optimal MATH workflow emerged from 5-parent fusion at iteration 15, combining nodes $\{1, 9, 2, 3, 4\}$ with scores $\{44.5\%, 42.9\%, 46.2\%, 50.4\%, 38.7\%\}$. Notably, this fusion surpassed all parents by a substantial margin (+5.3% over the best parent), demonstrating DAGO's ability to synthesize complementary problem-solving strategies.

DAGO-discovered MATH workflow (node_15)

```python
class Workflow:
    async def __call__(self, problem: str, **kwargs) -> str:
        # Step 1: Generate initial solution with detailed steps
        initial = await self.custom(input=problem,
            instruction=self.prompts.SOLVE_PROMPT)

        # Step 2: Verify the initial solution in the problem context
        verify = await self.programmer(
            problem=f"Problem: {problem}\n"
                    f"Initial solution: {initial['response']}",
            analysis="Extract and verify numerical calculations")

        # Step 3: Condition alternatives on the solution and feedback
        context = (
            f"Problem: {problem}\n"
            f"Initial solution: {initial['response']}\n"
            f"Verification feedback: {verify['output']}"
        )
        alts = []
        for _ in range(2):
            alt = await self.custom(input=context,
                instruction=self.prompts.ALT_SOLVE_PROMPT)
            alts.append(alt['response'])

        # Step 4: Ensemble for consistency
        final = await self.sc_ensemble(
            solutions=[initial['response']] + alts, problem=problem)
        return final['response']
```

Associated Prompts. The workflow employs two carefully designed prompts that emerged through multi-parent fusion:

MATH workflow prompts (node_15)

```
SOLVE_PROMPT = """Solve this math problem step by step:
1. Break down the problem into clear steps
2. Show all work and calculations
3. Simplify expressions fully - ALWAYS use fractions (\frac{a}{b})
   instead of decimals
4. Use proper mathematical notation (\sqrt{}, \pi, etc.)
5. Keep integers as integers
6. Put final answer in \boxed{}, simplified. Use ONE \boxed{}."""

ALT_SOLVE_PROMPT = """Solve this math problem using a different
approach if possible:
1. Try an alternative solution method
2. Show complete work
3. Verify calculations
4. Simplify completely - ALWAYS use fractions (\frac{a}{b})
   instead of decimals
5. Keep integers as integers
6. Put final answer in \boxed{}, simplified. Use ONE \boxed{}."""
```

Prompt Design Insights: The prompts explicitly enforce fraction notation over decimals, addressing a common failure mode where LLMs output decimal approximations that fail exact-match evaluation. The dual-prompt strategy (primary solve + alternative approach) encourages methodological diversity while maintaining consistent output formatting.

Multi-Parent Fusion Analysis. Examining the parent workflows reveals how fusion synthesized complementary strategies:

- **Node 1** (44.5%): Decomposition-based approach with problem breakdown and computational verification.
- **Node 9** (42.9%): Expert-style solving with programmatic verification and format post-processing.
- **Node 2** (46.2%): Multi-approach generation with iterative solution exploration.
- **Node 3** (50.4%): Solve-verify-ensemble with alternative solution generation (highest-scoring parent).
- **Node 4** (38.7%): Direct computational approach via Programmer operator.

The optimal workflow inherited the ensemble mechanism from Node 3, the computational verification from Nodes 1 and 9, and the multi-solution diversity strategy from Node 2. This combination enables robust error detection through cross-validation while maintaining solution diversity.

Comparison with AFlow MATH Workflow. AFlow's published MATH workflow employs a different strategy:

AFlow MATH workflow (from published results)

```
# AFlow MATH: Programmer + 4x Custom refinement + ScEnsemble
prog = await Programmer(problem=problem)
custom1 = await Custom(input=problem, instruction=REFINE_PROMPT)
custom2 = await Custom(input=problem, instruction=DETAILED_PROMPT)
custom3 = await Custom(input=problem, instruction=ALT_PROMPT_1)
custom4 = await Custom(input=problem, instruction=ALT_PROMPT_2)
final = await ScEnsemble([prog, custom1, custom2, custom3, custom4])
```

Key Structural Differences:

1. **Verification Integration**: DAGO verifies the initial solution in the context of the original problem and passes the resulting feedback to the alternative-solution stage.
2. **Solution Coupling**: Each alternative receives the original problem, the initial solution, and the verification feedback. The final ensemble combines the initial solution with these feedback-conditioned alternatives, whereas the displayed AFlow workflow generates its candidates without these inter-candidate dependencies.
3. **Operator Invocations**: The displayed DAGO workflow uses five high-level operator invocations (one solve, one verification, two alternatives, and one ensemble), compared with six in the displayed AFlow workflow. These counts refer to workflow-level operators, not the number of underlying model or tool calls within each operator.

F.3 Mathematical Reasoning: GSM8K Dataset

Best GSM8K Fusion Workflow (Node 11, Validation Score: 95.8%). GSM8K represents a case where the initial population already achieved near-optimal performance (95.8%), and fusion successfully maintained this performance through intelligent recombination. Node 11 emerged from parents $\{4, 1, 0, 7, 6\}$ at iteration 11.

DAGO-discovered GSM8K workflow (node_11)

```
class Workflow:
    async def __call__(self, problem: str, **kwargs) -> str:
        # Multi-strategy parallel generation
        direct = await self.custom(input=problem,
            instruction=self.prompts.DIRECT_SOLVE)   # Reasoning-based
        math = await self.custom(input=problem,
            instruction=self.prompts.MATH_SOLVE)     # Mathematical approach
        code = await self.programmer(problem=problem)  # Programmatic

        # Self-consistency ensemble
        solutions = [direct['response'], math['response'],
                     code.get('output', '')]
        final = await self.sc_ensemble(solutions=solutions, problem=problem)
        return final['response']
```

Associated Prompts. The workflow employs two complementary prompts targeting different reasoning styles:

GSM8K workflow prompts (node_11)

```
DIRECT_SOLVE = """Solve this math word problem using clear logical
reasoning:
1. Read the problem carefully
2. Break down the key information
```

```
3. Solve step by step using plain language
4. Double check your calculations
5. State ONLY the final numerical answer. No units, no words,
   no punctuation. Just the number."""

MATH_SOLVE = """Solve this math word problem using mathematical
equations:
1. Identify the variables and unknowns
2. Write and solve equations
3. Show your work clearly
4. Verify your solution
5. State ONLY the final numerical answer. No units, no words,
   no punctuation. Just the number."""
```

Prompt Design Insights: The dual-prompt strategy exploits complementary reasoning modes: DIRECT_SOLVE encourages intuitive step-by-step reasoning, while MATH_SOLVE promotes formal algebraic manipulation. Combined with the Programmer operator's computational verification, this creates a three-way ensemble that captures diverse solution paths. The strict output format ("Just the number") addresses GSM8K's exact-match evaluation requirements.

Convergence Analysis. The tied scores between nodes 11 and 19 (both 95.8%) reveal an important pattern: multiple structurally different workflows can achieve equivalent performance on well-structured mathematical reasoning tasks. This suggests that GSM8K problems have clear solution paths that multiple reasonable strategies can exploit. The key insight is that DAGO's exploration discovered this equivalence class while maintaining diversity, whereas single-parent methods might converge to a single solution path.

F.4 Code Generation: HumanEval Dataset

Best HumanEval Fusion Workflow (Node 12, Validation Score: 88.9%). The optimal HumanEval workflow emerged from 5-parent fusion at iteration 12, combining nodes $\{9, 0, 7, 8, 11\}$. This workflow introduces a sophisticated *conditional retry* mechanism that distinguishes it from simpler approaches.

DAGO-discovered HumanEval workflow (node_12)

```python
class Workflow:
    async def __call__(self, problem: str, **kwargs) -> str:
        entry_point = kwargs.get('entry_point', '')

        # 1. Fast path: try simple solution first
        result = await self.code_gen(problem=problem,
            entry_point=entry_point,
            instruction=f"Function MUST be named '{entry_point}'. "
                        f"{self.prompts.INITIAL_SOLVE}")
        test_result = await self.test(problem, result['response'], entry_point)

        if test_result.get('result', ''):
            return test_result.get('solution', '')  # Early exit on success

        # 2. Slow path: analyze failure and retry with diversity
        analysis = await self.custom(
            input=problem + "\n\nError in previous solution.",
            instruction=self.prompts.ANALYZE_PROBLEM)

        # 3. Generate multiple informed solutions
        solutions = []
        for _ in range(3):
            result = await self.code_gen(
                problem=problem + "\n\nAnalysis: " + analysis['response'],
                entry_point=entry_point,
                instruction=f"Function MUST be named '{entry_point}'. "
                            f"{self.prompts.CAREFUL_SOLVE}")
            solutions.append(result['response'])

        # 4. Ensemble and final test
        final = await self.ensemble(solutions=solutions, problem=problem)
        test_result = await self.test(problem, final['response'], entry_point)
        return test_result.get('solution', '')
```

Associated Prompts. The workflow employs three prompts for different stages of the adaptive solving process:

HumanEval workflow prompts (node_12)

```
INITIAL_SOLVE = """Write a Python function that solves the given problem.
Key requirements:
- Return type MUST match docstring specification EXACTLY
  (e.g. "Yes"/"No" vs True/False)
- Preserve original docstring exactly as given
- Handle all edge cases (empty input, negative numbers, zeros)
- Include helper functions inside main function if needed
- Pre-loaded modules (no import needed): math, re, hashlib,
  List, Dict, Tuple, Optional, Any

Output ONLY raw Python code starting with 'def'.
Do NOT use ``` code blocks. No markdown, no explanations."""

ANALYZE_PROBLEM = """Analyze the problem requirements and potential
edge cases:
1. What exact return type is required by the docstring?
2. What are the key input edge cases to handle?
3. What are potential algorithmic challenges?

Output ONLY the analysis points in clear, concise bullet points."""

CAREFUL_SOLVE = """Write a careful implementation considering the
analysis provided.
Requirements:
- Return type MUST match docstring specification EXACTLY
- Preserve original docstring exactly as given
- Handle ALL identified edge cases
- Pre-loaded modules (no import needed): math, re, hashlib,
  List, Dict, Tuple, Optional, Any
- Optimize for correctness over cleverness
- Test your logic step by step

Output ONLY raw Python code starting with 'def'."""
```

Prompt Design Insights: The three-stage prompt design reflects the adaptive complexity pattern: INITIAL_SOLVE attempts a direct solution, ANALYZE_PROBLEM performs failure diagnosis, and CAREFUL_SOLVE generates informed retry attempts. The explicit emphasis on return type matching ("MUST match docstring specification EXACTLY") addresses a critical HumanEval failure mode where semantically correct solutions fail due to type mismatches (e.g., returning `True` instead of `"Yes"`).

Emergent Design Pattern: Conditional Complexity. This workflow exhibits an emergent *adaptive complexity* pattern: simple problems take the fast path (2 LLM calls), while complex problems trigger the full analysis-retry mechanism (7 LLM calls). This pattern was not explicitly designed but emerged through multi-parent fusion—Node 9 contributed the early-exit mechanism, while Node 11 contributed the analysis-based retry logic.

F.5 Code Generation: MBPP Dataset

Best MBPP Workflow (Node 23, Validation Score: 86.0%). The optimal MBPP workflow emerged from parents $\{8, 6, 16, 13, 11\}$ at iteration 23, achieving 86.0% Pass@1.

DAGO-discovered MBPP workflow (node_23)

```python
class Workflow:
    async def __call__(self, problem: str, **kwargs) -> str:
        entry_point = kwargs.get('entry_point', '')

        # 1. Analyze problem into subtasks
        analysis = await self.custom(
            input=problem, instruction=self.prompts.ANALYZE_PROMPT)

        # 2. Generate candidate implementations (guided by analysis)
        solutions = []
```

```
        for _ in range(3):
            result = await self.code_gen(
                problem=analysis['response'],
                entry_point=entry_point,
                instruction=f"Function name MUST be '{entry_point}'. "
                            "Output ONLY raw Python code starting with 'def'.")
            solutions.append(result['response'])

        # 3. Ensemble to select a well-formed best candidate
        best = await self.ensemble(
            solutions=solutions,
            problem=f"Select the best solution that starts with 'def {entry_point}'. "
                    "Output raw Python code only.")

        # 4. Test and refine up to 2 attempts
        solution = best['response']
        for _ in range(2):
            test_result = await self.test(problem, solution, entry_point)
            if test_result.get('result', ''):
                return test_result.get('solution', '')
            solution = test_result.get('solution', solution)
        return solution
```

Associated Prompts. The MBPP workflow uses a structured analysis prompt that guides problem decomposition:

MBPP workflow prompt (node_23)

```
ANALYZE_PROMPT = """Break down this MBPP programming problem into clear subtasks:
1. First identify the input parameters and return type requirements
2. List the key algorithmic steps needed
3. Note any edge cases to handle
4. Identify any helper functions needed

Focus on Python implementation details and test case handling.
Provide your analysis as clear text that will guide implementation.
Do not include any code in this analysis step."""
```

Prompt Design Insights: The analysis prompt enforces an explicit decomposition (I/O requirements, algorithmic steps, edge cases) before code synthesis. This shared analysis provides structured guidance for all candidate solutions, improving robustness without eliminating implementation diversity.

Comparison with AFlow MBPP Workflow. AFlow's published MBPP workflow uses a simpler generate-test-fix pattern:

AFlow MBPP workflow (from published results)

```
# AFlow MBPP: 3x CustomCodeGenerate + ScEnsemble + Test (+ Custom fix)
solutions = []
for _ in range(3):
    sol = await CustomCodeGenerate(problem=problem, entry_point=entry_point,
        instruction=CODE_GENERATE_PROMPT)
    solutions.append(sol['response'])
best = await ScEnsemble(solutions=solutions, problem=problem)
test_result = await Test(problem=problem, solution=best['response'],
    entry_point=entry_point)
if not test_result['result']:
    fixed = await Custom(input=f"Problem: {problem}\nFailed solution:
    ↪  {best['response']}\nError: {test_result['solution']}",
        instruction=FIX_CODE_PROMPT)
    return fixed['response']
return test_result['solution']
```

Key Differences:

- **Problem Analysis**: DAGO explicitly analyzes problems before code generation, providing semantic guidance. AFlow generates code directly without structured analysis.

- **Information Flow**: DAGO shares a single analysis across all candidates by feeding it into each code-generation call, enabling informed diversity.
- **Repair Loop**: DAGO performs up to two test-refine iterations using the Test operator feedback, whereas AFlow uses a single test followed by a prompt-based fix when needed.

F.6 Question Answering: HotpotQA Dataset

Best HotpotQA Workflow (Node 29, Validation Score: 79.3%). HotpotQA presents unique challenges due to multi-hop reasoning requirements. The optimal workflow emerged from parents $\{19, 16, 5, 14, 27\}$ at iteration 29.

DAGO-discovered HotpotQA workflow (node_29)

```python
class Workflow:
    async def __call__(self, problem: str, **kwargs) -> str:
        # 1. Extract structured facts with multi-hop awareness
        facts = await self.custom(input=problem,
            instruction=self.prompts.EXTRACT_FACTS)

        # 2. Generate answer candidates using extracted facts
        candidates = []
        for _ in range(3):
            answer = await self.custom(
                input=f"{problem}\nExtracted Facts: {facts['response']}",
                instruction=self.prompts.GENERATE_ANSWER)
            candidates.append(answer['response'])

        # 3. Format-aware ensemble
        final = await self.custom(input="\n".join(candidates),
            instruction=self.prompts.ENSEMBLE_FORMAT)
        return final['response']
```

Associated Prompts. The HotpotQA workflow employs three sophisticated prompts that address multi-hop reasoning and format-sensitive evaluation:

HotpotQA workflow prompts (node_29)

```
EXTRACT_FACTS = """Extract only the most relevant facts from all
contexts that directly relate to answering the question.
CRITICAL: This is a multi-hop question requiring connecting facts
across passages.

Rules for extraction:
1. Copy EXACT words and phrases from the passages - do not
   paraphrase or use synonyms
2. For numbers: Copy the exact format (word vs digit) as written
   in the passage
3. Include facts needed for BOTH hops of reasoning
4. For yes/no questions: Include the exact facts that confirm/deny

Format: List key facts in bullet points, showing connections:
- First hop facts: [Facts needed for first reasoning step]
- Second hop facts: [Facts needed for second reasoning step]
- Connection: [How the facts link together]

Output ONLY the relevant facts and connections, no explanations."""

GENERATE_ANSWER = """Based on the extracted facts, determine the
precise answer.
CRITICAL RULES:
1. Use ONLY exact words copied from the passage - never use synonyms
2. Match number format exactly as written:
   - If passage says "six", output "six" (not "6")
   - If passage says "6", output "6" (not "six")
3. For yes/no questions: output only lowercase "yes" or "no"
4. Remove articles (a/an/the) unless part of proper name
5. Keep it brief - use only essential words (1-5 words maximum)
6. Never output full sentences or explanations

Output ONLY the exact answer phrase from the passage (1-5 words)."""

ENSEMBLE_FORMAT = """Select the best answer from the candidates that
```

```
strictly follows these format rules:
1. Uses EXACT words from passage (no synonyms)
2. Matches number format from passage exactly
3. For yes/no: only lowercase "yes" or "no"
4. No articles (a/an/the) unless part of proper name
5. 1-5 words maximum, no sentences

CRITICAL: Score each candidate (0-10):
- Subtract 10 if number format doesn't match passage exactly
- Subtract 10 if it uses synonyms instead of exact words
- Subtract 10 if yes/no answer isn't lowercase
- Subtract 5 if it contains unnecessary articles
- Subtract 2 points per word beyond 5 words

Select the candidate with highest score.
Output ONLY the exact answer phrase (1-5 words)."""
```

Prompt Design Insights: This prompt suite represents the most sophisticated format-aware design discovered by DAGO. The EXTRACT_FACTS prompt explicitly structures multi-hop reasoning with "First hop / Second hop / Connection" format. The GENERATE_ANSWER prompt includes concrete examples of number format matching ("six" vs "6"), addressing a critical HotpotQA evaluation quirk. The ENSEMBLE_FORMAT prompt introduces a novel *scoring rubric* that guides the LLM to evaluate candidates against format criteria, effectively implementing learned evaluation heuristics within the prompt itself.

Comparison with AFlow HotpotQA Workflow. AFlow's published HotpotQA workflow:

AFlow HotpotQA workflow (from published results)

```python
# AFlow HotpotQA: 3x AnswerGenerate + ScEnsemble + Custom format
ans1 = await AnswerGenerate(problem=problem)
ans2 = await AnswerGenerate(problem=problem)
ans3 = await AnswerGenerate(problem=problem)
best = await ScEnsemble([ans1, ans2, ans3], problem=problem)
final = await Custom(input=best, instruction=FORMAT_PROMPT)
```

Key Insight: DAGO's workflow introduces an explicit *fact extraction* step that addresses HotpotQA's multi-hop nature. This structural difference enables the model to reason about fact connections before answer generation, whereas AFlow's parallel generation lacks this grounded reasoning step.

F.7 Question Answering: DROP Dataset

Best DROP Fusion Workflow (Node 25, Validation Score: 84.0%). DROP requires discrete reasoning over paragraphs. The optimal workflow emerged from parents $\{23, 14, 8, 9, 17\}$ at iteration 25.

DAGO-discovered DROP workflow (node_25)

```python
class Workflow:
    async def __call__(self, problem: str, **kwargs) -> str:
        # 1. Identify question type (counting, comparison, arithmetic, etc.)
        type_result = await self.custom(input=problem,
            instruction=self.prompts.TYPE_PROMPT)
        question_type = type_result['response']

        # 2. Type-aware solving
        answer = await self.custom(
            input=f"{problem}\nQuestion Type: {question_type}",
            instruction=self.prompts.SOLVE_PROMPT)

        # 3. Format answer according to DROP requirements
        final = await self.custom(input=answer['response'],
            instruction=self.prompts.FORMAT_PROMPT)
        return final['response']
```

Associated Prompts. The DROP workflow employs three prompts implementing a type-aware solving pipeline:

DROP workflow prompts (node_25)

```
TYPE_PROMPT = """Identify the question type from the following
categories:
1. Numerical calculation (how many more/less, difference between)
2. Direct number extraction (how many, what number)
3. Text span extraction (who, what, where)
4. Date/time calculation
5. Counting entities

Output ONLY the category number (1-5)."""

SOLVE_PROMPT = """Answer the question based on the provided context
and question type.
Follow these rules:
1. For calculations: Show work, then give final number
2. For text spans: Extract exact words from passage
3. For counting: Count carefully and verify
4. For dates: Calculate precisely
5. If multiple valid answers exist, separate them with |

Output format: First show your work, then "ANSWER:" followed by
the answer."""

FORMAT_PROMPT = """Format the answer following these rules:

1. For numerical answers:
   - Use only Arabic numerals (1, 2, 3)
   - Remove all units and words
   - For differences/comparisons, output only the final number
   - If multiple valid numbers exist, separate with |

2. For text answers:
   - Extract the exact span from passage
   - Remove articles (the, a, an) unless part of proper name
   - Remove trailing punctuation
   - If multiple valid spans exist, separate with |

Examples of CORRECT format:
- Numbers: "3", "74.6", "12", "3|4"
- Text: "Timberwolves", "the Federales", "red car", "dog|cat"

Examples of INCORRECT format:
- Numbers: "Three", "74.6%", "3 years", "about 12"
- Text: "The Timberwolves", "Timberwolves.", "they won the game"

Output ONLY the answer. For numbers: Arabic numerals only.
For text: exact span from passage. No sentences, no units."""
```

Prompt Design Insights: The DROP prompts demonstrate the most explicit type-aware design pattern. The TYPE_PROMPT creates a 5-way classification that routes to specialized solving strategies. The FORMAT_PROMPT includes both positive and negative examples—a prompt engineering technique that emerged through fusion and significantly improves format compliance. The explicit handling of multiple valid answers ("separate with |") addresses DROP's multi-answer evaluation protocol.

Emergent Question Typing. A notable discovery is the explicit *question type classification* step. The TYPE_PROMPT categorizes questions into: counting, arithmetic, comparison, span extraction, and date/time reasoning. This type-aware approach enables specialized solving strategies per question category, improving accuracy on DROP's diverse question types.

Initial vs. Fusion Performance. On DROP, the best initial workflow (Node 9) scores 82.9% on the validation set, while the best fusion workflow (Node 25) reaches 84.0%. This is an improvement of 1.1 percentage points, or approximately 1.3% relative to the best initial score. Node 25 therefore also has the highest stored validation score in the archive and is selected by the return rule in Algorithm 1. These validation scores are distinct from the final test-set results.

F.8 Cross-Task Pattern Analysis

Analyzing the six optimal workflows reveals several emergent design patterns that DAGO consistently discovered through multi-parent fusion:

Table 12: Emergent workflow patterns discovered by DAGO across task types. ✓ indicates pattern presence.

Pattern	GSM8K	MATH	HumanEval	MBPP	HotpotQA	DROP
Multi-Solution Ensemble	✓	✓	✓	✓	✓	–
Computational Verification	✓	✓	✓	✓	–	–
Conditional Retry	–	–	✓	✓	–	–
Explicit Problem Analysis	–	–	✓	✓	✓	✓
Type-Aware Processing	–	–	–	–	–	✓
Format-Aware Generation	–	✓	✓	✓	✓	✓

Pattern 1: Task-Appropriate Ensemble Sizes. Mathematical tasks (GSM8K, MATH) and code generation (HumanEval, MBPP) benefit from 3-way ensembles, while question answering tasks show mixed results. This suggests that ensemble diversity has diminishing returns when answer formats are highly constrained (as in DROP).

Pattern 2: Verification Operator Usage. The Programmer operator appears exclusively in mathematical reasoning workflows, providing computational verification. Code generation tasks use the Test operator instead. Question answering tasks eschew computational verification entirely, relying on prompt-based consistency checks.

Pattern 3: Conditional Complexity. Code generation tasks (HumanEval, MBPP) developed conditional retry mechanisms that adapt workflow complexity to problem difficulty. This pattern did not emerge in other task types, suggesting it is specific to tasks where quick feedback (test results) is available.

F.9 Multi-Parent Fusion Depth Analysis

To understand how multi-parent fusion contributes to workflow quality, we analyze the ancestral depth of optimal workflows:

Table 13: Ancestral analysis of best fusion workflows. Unique ancestors counts distinct nodes in the ancestral DAG.

Dataset	Best Node	Direct Parents	Unique Ancestors	Max Depth
GSM8K	11	5	5	1
MATH	15	5	5	1
HumanEval	12	5	7	2
MBPP	23	5	13	3
HotpotQA	29	5	21	7
DROP	25	5	18	7

Key Insight: Fusion Depth Reflects the Hierarchical Complexity of Optimal Workflow Structures. We observe substantial variation in the *structural depth* of optimal workflows across tasks. For GSM8K, MATH, and HumanEval, although the underlying tasks can be highly challenging (particularly MATH), their optimal workflows exhibit relatively *flat* structures, primarily composed of parallel reasoning, verification, and ensemble components. As a result, their best workflows have shallow ancestries (max depth ≤ 2), and a small number of fusion generations is sufficient to recombine key functional modules. In contrast, MBPP, HotpotQA, and DROP require workflows with pronounced *hierarchical and multi-stage* structures, where intermediate abstractions—such as explicit problem analysis, fact extraction, or question type identification—must be constructed before

conditional solving and result aggregation. Such hierarchical compositions are difficult to synthesize via a single fusion step, and instead benefit from *multi-generation fusion*, where intermediate modules are progressively refined and recombined across successive evolutionary stages. These findings suggest that the primary role of multi-parent fusion is not merely to accelerate search, but to enable the *hierarchical evolution of workflow structures*, allowing complex reasoning programs to emerge through modular reuse and cross-generational composition.

G Full Experimental Results

This section provides comprehensive experimental data supporting the main results presented in Section 5. We report detailed iteration-by-iteration performance, optimization dynamics, and statistical analyses across all six benchmarks.

G.1 Optimization Summary

We summarize the key optimization outcomes across all six benchmarks. Four benchmarks met the recent-score stopping criterion before the iteration limit, whereas HotpotQA and DROP completed all 30 fusion rounds.

G.2 Detailed Optimization Statistics

Table 14 provides comprehensive statistics capturing the optimization dynamics for each benchmark.

Table 14: DAGO optimization statistics per benchmark. Initial scores use one validation execution; fusion scores average three. Reward-distribution statistics are computed over fusion nodes only, and Δ is measured in percentage points.

Statistic	**GSM8K**	**MATH**	**HumanEval**	**MBPP**	**HotpotQA**	**DROP**
Optimization Configuration						
Total Iterations	28	27	25	25	30	30
Early Stopped	Yes	Yes	Yes	Yes	No	No
Stopping Trigger	Stagnation	Stagnation	Stagnation	Stagnation	Max Iter	Max Iter
DAG Statistics						
Total Nodes	38	37	35	35	40	40
Initial Nodes	10	10	10	10	10	10
Fusion Nodes	28	27	25	25	30	30
Total Edges	140	135	125	125	150	150
Leaf Nodes	11	9	6	11	11	7
Performance Statistics						
Best Init Score	95.8%	50.4%	87.9%	83.7%	77.2%	82.9%
Best Fusion Score	95.8%	55.7%	88.9%	86.0%	79.3%	84.0%
Best Overall	95.8%	55.7%	88.9%	86.0%	79.3%	84.0%
Δ (points)	0.0	+5.3	+1.0	+2.3	+2.1	+1.1
Fusion Reward Distribution						
Mean Reward	93.9%	49.6%	86.0%	78.4%	70.8%	77.5%
Std Dev	1.0%	2.7%	2.3%	16.3%	12.0%	6.0%
Min Reward	92.2%	43.4%	78.8%	0.0%	21.0%	59.0%
Max Reward	95.8%	55.7%	88.9%	86.0%	79.3%	84.0%

Observed Stopping Outcomes. With a five-round recent-score window, stopping tolerance $\delta_{\text{stop}} = 10^{-4}$, and a minimum of 20 fusion rounds, HumanEval and MBPP stopped after 25 rounds, MATH after 27, and GSM8K after 28. HotpotQA and DROP reached the 30-round limit. These counts describe the recorded stopping outcomes; the highest-scoring workflow is retained independently of the recent-score stopping decision.

Failure Mode Analysis. MBPP exhibits the highest variance ($\sigma = 16.3\%$) and contains one complete failure (Node 20, score=0.0%), driven by an over-aggressive output "sanitization" step that prepends `def` to non-`def` outputs, often producing invalid Python code for unit-test evaluation. HotpotQA also contains a 0.0-scoring initial workflow (Node 2) that fails at runtime due to a missing prompt import. These outliers highlight the need for robust runtime and format checks in LLM-based optimization. Importantly, the DAG structure isolates such failures, and their low rewards make them unlikely to be selected as parents in later fusions.

G.3 Score Distribution Analysis

Table 15 presents detailed percentile analysis of workflow scores across all nodes.

Table 15: Percentiles of stored validation scores across all nodes (initial + fusion). Initial scores use one validation execution; fusion scores average three.

Percentile	GSM8K	MATH	HumanEval	MBPP	HotpotQA	DROP
Min (P0)	29.5%	38.7%	66.7%	0.0%	0.0%	34.6%
P25	92.8%	46.2%	84.8%	77.9%	69.2%	74.4%
Median (P50)	93.8%	48.7%	86.9%	81.0%	74.3%	79.0%
P75	94.3%	51.0%	87.9%	83.3%	75.3%	80.9%
Max (P100)	95.8%	55.7%	88.9%	86.0%	79.3%	84.0%

Distribution Characteristics. GSM8K shows a highly concentrated distribution (IQR=1.4%) with most workflows performing near optimally, reflecting the task's relatively straightforward nature. In contrast, MBPP and HotpotQA exhibit heavy tails with rare near-zero outliers. For MBPP, this arises from a failed fusion workflow (Node 20), while for HotpotQA it stems from an invalid initial workflow (Node 2); both amplify the lower tail while the bulk of nodes remain clustered in the high-performance regime.

G.4 Best Workflow Ancestry Analysis

Table 16 provides detailed lineage information for each optimal workflow, tracing the contribution paths through the DAG.

Table 16: Ancestry analysis of best-performing workflows per benchmark.

Dataset	Best Node	Parent IDs	Parent Scores	Unique Ancestors	Max Depth
GSM8K	11	{4,1,0,7,6}	{95.8,83.3,94.7,94.3,94.3}%	5	1
MATH	15	{1,9,2,3,4}	{44.5,42.9,46.2,50.4,38.7}%	5	1
HumanEval	12	{9,0,7,8,11}	{87.9,87.9,87.9,87.9,86.9}%	7	2
MBPP	23	{8,6,16,13,11}	{77.9,76.7,81.0,81.0,80.2}%	13	3
HotpotQA	29	{19,16,5,14,27}	{75.1,75.3,74.5,73.3,74.9}%	21	7
DROP	25	{23,14,8,9,17}	{79.3,78.5,34.6,82.9,75.6}%	18	7

Direct-Parent Workflow and Prompt Contributions. To make the lineage in Table 16 more actionable, we inspect the *direct parents* of each best node and summarize the reusable modules and prompt constraints that are preserved in the child.

- **GSM8K (Node 11 from {4,1,0,7,6}).** The highest-scoring parent (Node 4) already implements a three-branch ensemble (direct reasoning, equation-based reasoning, and a programmatic `Programmer` branch), followed by `ScEnsemble`. The remaining parents contribute complementary "interface discipline": explicit examples and strict "output-only-a-number" constraints (Node 0/7), plus a verification-style prompt that forces a binary correctness judgment (Node 6). The fused child retains the tri-branch ensemble structure while strengthening format constraints on the programmatic branch and final selection prompt, reducing exact-match failures driven by stray units or punctuation.

- **MATH (Node 15 from {1,9,2,3,4}).** Parent workflows specialize along different dimensions: a solve–verify–multi-solution–ensemble backbone (Node 3), multi-approach generation (Node 2), problem-type routing with strict fraction/$\boxed{\cdot}$ formatting rules (Node 4), decomposition plus computational cross-checking (Node 1), and an explicit final answer extractor that outputs only a single boxed expression (Node 9). The child workflow keeps the backbone from Node 3, but imports the exactness constraints emphasized across Nodes 2/4/9 (e.g., enforcing fractions over decimals and a single $\boxed{\cdot}$), and propagates these constraints into both the verification instruction and the prompts used for alternative solutions.
- **HumanEval (Node 12 from {9,0,7,8,11}).** The parent set decomposes into three recurring patterns: (i) a baseline generate–test loop (Node 0), (ii) multi-solution generation with `ScEnsemble` before testing (Node 7), and (iii) test-driven conditional retry with an explicit analysis step followed by three re-generations (Nodes 9 and 11). Node 8 adds an explicit requirement-parsing step that distills edge cases and I/O constraints. The fused workflow adopts the conditional retry structure (fast path with early exit; slow path with analysis-driven re-generation) while tightening prompt constraints that are repeatedly highlighted by its parents, including strict `entry_point` naming and return-type/docstring fidelity ("MUST match docstring specification EXACTLY").
- **MBPP (Node 23 from {8,6,16,13,11}).** The parents span diverse code-synthesis heuristics: example retrieval as in-context priming (Node 6), generate–test–select among candidates (Node 8), augmentation via synthesized unit tests (Node 11), iterative fix loops with early stopping and ensemble fallback (Node 13), and an analysis-first design where the analysis itself is requested as executable code and then fed into generation (Node 16). The child retains the analysis-first "shared guidance" idea and the multi-candidate strategy (Node 16/8), but relaxes the over-constrained "analysis-as-code" requirement into a plain-text decomposition while strengthening the code interface (raw Python starting with `def`) and adding a lightweight test-refine loop (two iterations), improving robustness without collapsing diversity.
- **HotpotQA (Node 29 from {19,16,5,14,27}).** The parents represent increasingly format-aware QA pipelines: fact extraction followed by three candidate answers and ensembling (Nodes 5/16), a learned scoring rubric for candidate selection (Node 19 via `ENSEMBLE_FORMAT`), multi-stage refinement that explicitly corrects wording and formatting after an initial answer (Node 14), and deterministic post-processing heuristics (Node 27; lowercasing yes/no and article removal unless part of a proper name). The child workflow integrates these components by using an explicit multi-hop fact-extraction template (first hop/second hop/connection) and a rubric-based selector to penalize synonym/format violations, effectively combining grounding with stringent evaluation-aligned formatting.
- **DROP (Node 25 from {23,14,8,9,17}).** Node 9 contributes the core type-aware pipeline (question-type classification $\rightarrow$ type-conditioned solving $\rightarrow$ formatting). Nodes 14/17/23 contribute progressively stricter formatting prompts with positive/negative examples; Node 17 additionally introduces the `|` separator for multiple valid answers. Despite its low score, Node 8 supplies a normalization-oriented prompt that explicitly strips units/punctuation and enforces Arabic numerals, which is complementary to otherwise strong reasoning components. The fused child retains Node 9's routing structure but absorbs the strongest format-compliance constraints from Nodes 17 and 8, yielding a more robust formatting stage that better matches DROP's evaluation protocol.