Fusion is the New Mutation: Bandit-Guided Evolution on Workflow Graphs

Zhiwei Shang [1] **Jiahang Sun** [1] **Mingrong Gong** [2] **Mingze Kong** [1] **Zikun Qu** [1] **Pingchen Lu** [1] **Junhao Dong** [2] **Zhipiao Liu** [3] **Hongwei Yang** [3] **Guoqing Xie** [3] **Yao Shu** [4] **Zhongxiang Dai** [1]

Abstract

Agentic workflows, which orchestrate multiple large language model (LLM) calls, tools, and control flow, have become a powerful paradigm for complex reasoning tasks. Recent work formulates automated workflow discovery as a search problem in code space, but most existing methods rely on tree-based search with single-parent mutation, leading to structural isolation and inefficient exploration in highly discrete and compositional spaces. We propose **DAGO** (Directed Acyclic Graph Optimization), a principled framework that models workflow search as evolution over a directed acyclic graph with multi-parent fusion. DAGO maintains a shared DAG memory that enables lineage merging across search branches, and formulates parent selection as a contextual linear bandit problem. Using pretrained code embeddings and a Linear UCB policy, DAGO efficiently navigates the combinatorial fusion space. An LLM is then employed as a semantic fusion operator to synthesize new workflows by integrating complementary strengths from multiple parents. We evaluate DAGO on six benchmarks covering mathematical reasoning, code generation, and question answering. DAGO achieves state-of-the-art average performance while reducing token consumption. Ablation studies further confirm that multi-parent fusion and bandit-guided selection are key to its effectiveness.

[1]School of Data Science, The Chinese University of Hong Kong, Shenzhen, China [2]College of Computing and Data Science, Nanyang Technological University, Singapore [3]Huawei Technologies Co., Ltd., China [4]Artificial Intelligence Thrust, The Hong Kong University of Science and Technology (Guangzhou), China. Correspondence to: Zhongxiang Dai <daizhongxiang@cuhk.edu.cn>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea.

1. Introduction

The rapid advancement of large language models (LLMs) has fundamentally reshaped the paradigm for solving complex reasoning tasks. From early techniques such as Chain-of-Thought (CoT) prompting (Wei et al., 2022) to more sophisticated frameworks including ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023), the research focus has gradually shifted from single-shot prompt engineering toward the design of *agentic workflows* (Wang et al., 2024a; Hu et al., 2025; Zhang et al., 2025b; Niu et al., 2025; Zhang et al., 2025d). By orchestrating multiple LLM invocations, tool usages, and control-flow logic, agentic workflows enable structured problem solving for tasks such as mathematical reasoning and code generation, substantially outperforming single-pass inference approaches.

Despite their promise, designing effective agentic workflows remains highly challenging. Most existing approaches rely on expert-crafted workflows (Wu et al., 2024; Hong et al., 2024), which are labor-intensive and difficult to generalize across domains. To enable automation, recent methods such as ADAS (Hu et al., 2025) and AFlow (Zhang et al., 2025d) formulate workflow construction as a search problem in code space. In particular, AFlow significantly improves search efficiency by introducing Monte Carlo Tree Search (MCTS). However, these tree-based methods fundamentally inherit the paradigm of *single-parent mutation*: each new workflow node is derived from a single parent via local modifications, such as prompt edits or the insertion of additional steps.

We argue that this *unilineal mutation* paradigm is fundamentally inefficient for optimizing LLM-based workflows. Unlike continuous parameter spaces, the code space of workflows is highly discrete and combinatorial. High-performing workflows often integrate heterogeneous strengths; for example, one workflow may excel at problem decomposition, while another is particularly effective at code verification. Under a tree-structured search, such complementary structures discovered in different branches remain isolated and cannot be recombined through single-parent mutation. As a result, the search process is prone to local optima or requires prohibitively large sampling budgets to rediscover useful structures via random perturbations.

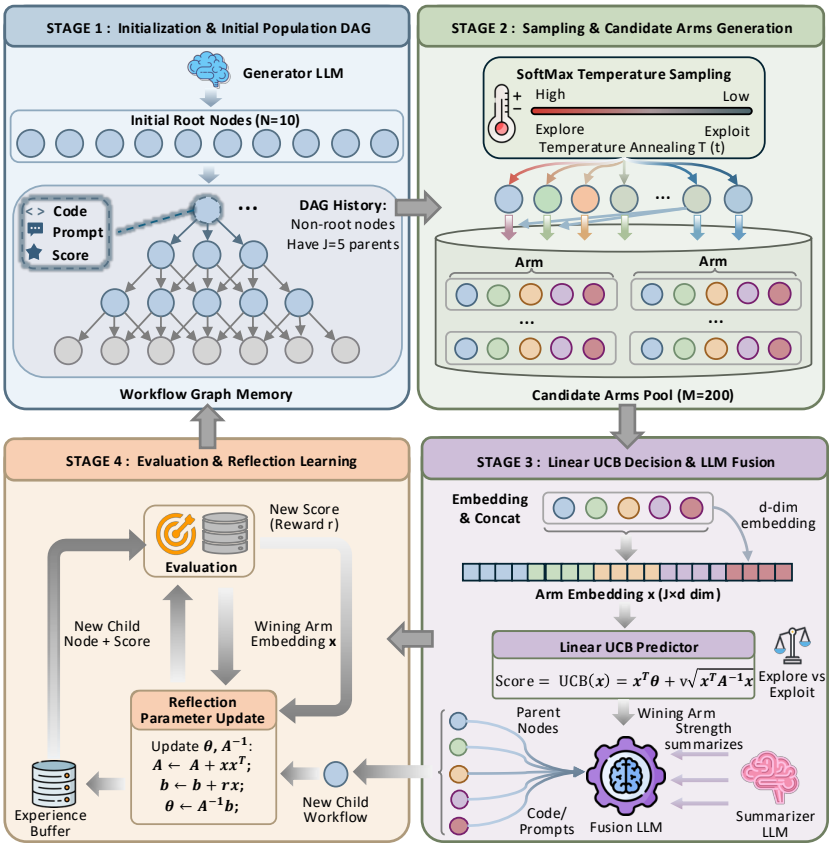


Figure 1. **Overview of the DAGO framework.** DAGO follows a four-stage iterative loop: (1) initialization with a DAG-structured workflow memory; (2) temperature-annealed sampling of candidate arms; (3) Linear-UCB-guided selection and LLM-based multi-parent fusion; and (4) evaluation and bandit update.

Motivated by these limitations, we propose a paradigm shift: in the evolution of agentic workflows, *fusion* should replace mutation as the primary driving force. Inspired by genetic recombination in evolutionary algorithms (Mitchell, 1998) and code merging practices in software engineering (Mens, 2002), we advocate *multi-parent fusion* as a systematic mechanism for synthesizing stronger workflows.

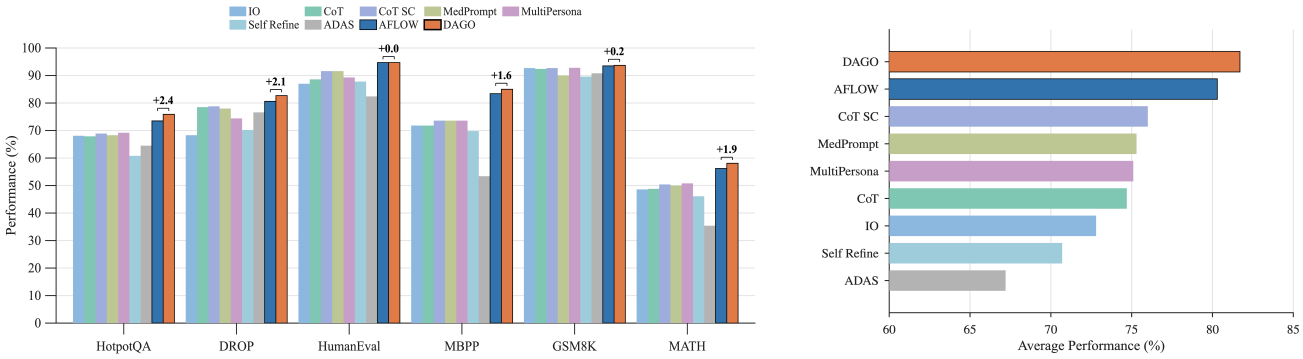
However, introducing fusion poses a new combinatorial challenge. Selecting J parents from a candidate pool of size N yields $\binom{N}{J}$ possible combinations, causing an exponential explosion of the search space. Randomly choosing parents is highly inefficient, while exhaustively enumerating all combinations is computationally infeasible. To address this core challenge, we propose **Directed Acyclic Graph Optimization (DAGO)**, a contextual bandit–guided framework for graph-based workflow evolution. DAGO introduces three key innovations.

1. **DAG-based memory**. We restructure the search history from a tree into a directed acyclic graph (DAG), allowing each workflow node to have multiple parents and explicitly tracking the lineage of structural improvements, thereby maximizing the reuse of historical exploration.

2. **LinUCB-guided structured selection**. We formulate parent selection as a contextual linear bandit problem. Using pretrained code embeddings as state representations, we adopt the Linear UCB algorithm (Li et al., 2010; Chu et al., 2011; Abbasi-yadkori et al., 2011) to balance exploration and exploitation, enabling the prediction of which parent combinations are likely to yield high-performing offspring in a vast combinatorial space.

3. **LLM-driven semantic fusion**. Instead of traditional token-level crossover, we leverage the semantic understanding capabilities of LLMs as fusion operators. By conditioning on parent workflows and their summarized advantages, the LLM resolves logical conflicts and synthesizes a new workflow that integrates complementary strengths.

We evaluate DAGO on six benchmarks spanning mathematical reasoning (MATH, GSM8K), code generation



(a) Performance comparison across benchmarks. *(b)* Method ranking across tasks.

Figure 2. **Overall performance across benchmarks and average ranking. (a)** Test performance on six benchmarks (3-run average; reported as Accuracy / Pass@1 / F1 depending on the task). Brackets annotate the absolute improvement of DAGO over AFlow on each benchmark. **(b)** Macro-average performance across all tasks, summarizing each method's overall ranking.

(HumanEval, MBPP), and question answering (HotpotQA, DROP). Experimental results demonstrate that DAGO consistently achieves state-of-the-art performance across all tasks, attaining an average accuracy of 81.7%, surpassing the previous best method AFlow by 1.4 percentage points while significantly reducing token consumption. Ablation studies further confirm that multi-parent fusion and bandit-guided selection are critical to these gains, validating the effectiveness of the principle that *fusion is the new mutation* for automated workflow design.

2. Related Work

Automated Search and Optimization of Agentic Workflows. Early agentic systems relied on expert-crafted orchestration, which is difficult to scale. Recent automation work broadly follows two lines. **Node- and instance-level optimization** improves prompts or orchestration *per query*: DSPy (Khattab et al., 2024), TextGrad (Yuksekgonul et al., 2025), and MIPRO (Opsahl-Ong et al., 2024) optimize prompting under fixed topologies, while AutoAgents (Chen et al., 2024), JudgeFlow (Ma et al., 2026), and RobustFlow (Xu et al., 2025) generate workflows on the fly; ScoreFlow (Wang et al., 2025) and FlowReasoner (Gao et al., 2025) directly emit workflow code via preference optimization. A concurrent anonymous submission (in the supplementary material) studies query-level orchestration via contextual-bandit selection from an evolved workflow pool. Despite their flexibility, *query-time orchestration/one-pass generation* incurs nontrivial inference cost and makes it hard to accumulate reusable, task-level structural knowledge. **Task-level structure search**, most related to our work, aims to discover reusable topologies for a task: GPTSwarm (Zhuge et al., 2024) and G-Designer (Zhang et al., 2025c) optimize multi-agent graphs, while ADAS (Hu et al., 2025), AFlow (Zhang et al., 2025d), EvoFlow (Zhang et al., 2025a), and MermaidFlow (Zheng et al., 2025) search executable code workflows. However, these methods largely follow *single-parent mutation* in tree-structured search, causing *structural isolation* that prevents recombining complementary structures across lineages.

Bandit Algorithms for Search and Structured Exploration in LLMs. Bandits provide a principled mechanism for exploration–exploitation in large action spaces and are increasingly used in LLM optimization (Xie et al., 2026). **Non-contextual bandits and MCTS.** MCTS is widely used for reasoning and planning (Zhou et al., 2023); in workflow search, AFlow adopts UCT to guide expansion (Zhang et al., 2025d). Yet UCT is effectively *context-free*, relying on counts and empirical rewards while ignoring workflow semantics, so estimates transfer poorly across similar workflows and exploration is sample-inefficient in vast code spaces. **Contextual bandits in LLM optimization.** Contextual bandits such as LinUCB are well-studied in recommendation (Li et al., 2010; Chu et al., 2011) and have been applied to prompt strategy selection (OPTS (Ashizawa et al., 2025), TRIPLE (Shi et al., 2024)), RAG module configuration (MBA-RAG (Tang et al., 2025)), and model routing (PILOT (Panda et al., 2025)). Most of these works select a *single* discrete action (one prompt/model). In contrast, our setting requires *combinatorial* selection of multiple parent workflows for fusion. We formulate this as a contextual linear bandit and apply Linear UCB over pretrained code embeddings to predict the fusion potential of parent combinations under sparse rewards.

3. Preliminary

Workflow Representation. We conceptualize an agentic workflow as a programmatic orchestration of LLM invocations, control logic, and data processing units. Formally, a workflow $w \in \mathcal{W}$ is defined as an executable Python class implementing an asynchronous `__call__` method.

This method maps a problem instance $x \in \mathcal{X}$ to a solution string $y \in \mathcal{Y}$ through a sequence of intermediate states. Following Zhang et al. (2025d), this code-based representation transcends static prompt templates, offering the Turing-complete expressiveness necessary for complex reasoning tasks while maintaining modularity and reproducibility.

Workflow Graph. The evolutionary trajectory of workflows is maintained in a DAG, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node $v_i \in \mathcal{V}$ represents a specific workflow instance, uniquely identified by the tuple $v_i = (\mathcal{C}_i, \mathcal{P}_i)$, where $\mathcal{C}_i$ denotes the structural code and $\mathcal{P}_i$ signifies the internal prompt set. Each node is associated with a performance metric $s_i \in [0, 1]$ and a semantic embedding vector $\mathbf{e}_i \in \mathbb{R}^d$ generated by a pre-trained encoder. An edge $(v_j, v_i) \in \mathcal{E}$ represents a lineage relationship, indicating that v_j served as one of the parental configurations in the fusion process that generated v_i. Unlike traditional tree-based search structures (e.g., MCTS), our DAG formulation explicitly accounts for multi-parent inheritance, enabling the recombination of diverse structural advantages.

Linear UCB. We formulate the selection of optimal parent nodes for fusion as a contextual multi-armed bandit problem. Let an *arm* $a = (v_1, \ldots, v_J) \in \mathcal{A}$ be an ordered subset of J nodes from $\mathcal{V}$. The contextual feature of arm a is constructed via a concatenation of parental embeddings: $\mathbf{x}_a = [\mathbf{e}_1; \mathbf{e}_2; \ldots; \mathbf{e}_J] \in \mathbb{R}^{Jd}$. To efficiently navigate the exploration-exploitation trade-off in this high-dimensional space, we employ Linear UCB. The algorithm maintains a weight vector $\boldsymbol{\theta} \in \mathbb{R}^{Jd}$ estimated via ridge regression. At each iteration, we select the arm a^* that maximizes the upper confidence bound:

$$\text{UCB}(\mathbf{x}_a) = \mathbf{x}_a^\top \boldsymbol{\theta} + \nu \sqrt{\mathbf{x}_a^\top \mathbf{A}^{-1} \mathbf{x}_a}, \tag{1}$$

where $\mathbf{A} = \sum \mathbf{x}\mathbf{x}^\top + \lambda \mathbf{I}$ is the covariance matrix capturing historical observations, λ is a regularization parameter, and ν modulates the exploration intensity. This mechanism ensures that the search prioritizes both empirically high-performing combinations and regions with high epistemic uncertainty.

4. Methodology

Building on the workflow/code representation and contextual bandit preliminaries in Section 3, we introduce **DAGO** (**D**irected **A**cyclic **G**raph **O**ptimization), an online evolutionary framework that optimizes agentic workflows via *multi-parent fusion* guided by principled exploration–exploitation control. DAGO maintains a persistent workflow memory as a DAG and iteratively expands it by (i) proposing multi-parent candidate arms, (ii) selecting one arm with Linear UCB, (iii) fusing the chosen parents into a child workflow using LLMs, and (iv) updating the bandit from the child's validation reward.

Workflow DAG Memory. DAGO stores all discovered workflows in a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (Section 3). Each node $v_i \in \mathcal{V}$ is an executable workflow with validation score s_i and a fixed semantic embedding $\mathbf{e}_i \in \mathbb{R}^d$ obtained by encoding its source code and internal prompts. Directed edges record lineage; unlike tree memories, the DAG supports *multi-parent* derivations, enabling improvements discovered in different branches to be recombined into a single descendant rather than remaining structurally isolated.

Candidate Arm Proposal via Annealed Score Sampling. At iteration t, DAGO forms a candidate set $\mathcal{A}_t$ of M arms, where each arm is an ordered tuple of J parent nodes. To bias proposals toward high-performing regions while preserving diversity, nodes are sampled from a temperature-controlled, score-based distribution. Specifically, we compute $w_i^{(t)} = \exp\left(\frac{s_i}{\tau_t}\right)$ and normalize it as $p_i^{(t)} = \frac{w_i^{(t)}}{\sum_{k \in \mathcal{V}} w_k^{(t)}}$. To ensure non-zero probability for every node, we apply a floor $p_{\min} > 0$ and renormalize $\tilde{p}_i^{(t)} = \frac{\max\left(p_i^{(t)}, p_{\min}\right)}{\sum_{k \in \mathcal{V}} \max\left(p_k^{(t)}, p_{\min}\right)}$. Here $\tau_t > 0$ is the sampling temperature, which we anneal linearly over the total budget T as $\tau_t = \tau_{\text{init}} + (\tau_{\text{final}} - \tau_{\text{init}})\frac{t}{T}$. Using $\tilde{p}^{(t)}$, we sample M candidate arms $\mathcal{A}_t = \{a_1, \ldots, a_M\}$, where each $a_m = (v_{m_1}, \ldots, v_{m_J})$ is formed by independently drawing J parents and then ordering them by descending validation score (to induce consistent positional semantics in later concatenation).

Linear UCB Arm Selection with Diagonal Uncertainty. Each arm a_m is mapped to a contextual feature by concatenating the embeddings of its ordered parents, $\mathbf{x}_m = [\mathbf{e}_{m_1}; \mathbf{e}_{m_2}; \ldots; \mathbf{e}_{m_J}] \in \mathbb{R}^{Jd}$. We assume a linear reward model for the (unknown) expected improvement of an arm, $\mathbb{E}[r_m \mid \mathbf{x}_m] = \mathbf{x}_m^\top \boldsymbol{\theta}$, with $\boldsymbol{\theta} \in \mathbb{R}^{Jd}$ learned online. Given the candidate set $\mathcal{A}_t$, DAGO selects the arm by the UCB criterion

$$a^* = \arg\max_{a_m \in \mathcal{A}_t} \left(\mathbf{x}_m^\top \boldsymbol{\theta} + \nu \sqrt{\mathbf{x}_m^\top \mathbf{A}^{-1} \mathbf{x}_m} \right), \tag{2}$$

where $\nu > 0$ controls exploration and $\mathbf{A} = \lambda \mathbf{I} + \sum_{(\mathbf{x}, r) \in \mathcal{B}} \mathbf{x}\mathbf{x}^\top$ is the (regularized) empirical covariance from past selected arms stored in buffer $\mathcal{B}$, with $\lambda > 0$. To keep per-iteration computation linear in Jd, we approximate $\mathbf{A}$ as diagonal, yielding

$$\mathbf{x}_m^\top \mathbf{A}^{-1} \mathbf{x}_m \approx \sum_{k=1}^{Jd} \frac{x_{m,k}^2}{A_{kk}}. \tag{3}$$

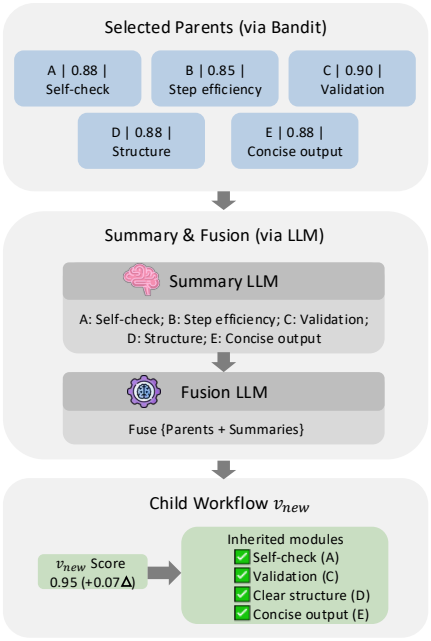


Figure 3. **One DAGO iteration (zoom-in of Stage 3).** LinUCB selects an ordered set of parent workflows; a summarization LLM distills each parent's core advantage; a fusion LLM synthesizes a child workflow conditioned on parents and summaries, typically inheriting complementary modules and improving the validation score.

LLM Multi-parent Fusion and Online Bandit Update. Given the selected arm $a^* = (v_1^*, \ldots, v_J^*)$, DAGO summarizes each parent workflow (code $\mathcal{C}_j$, prompts $\mathcal{P}_j$, score s_j) using a dedicated summarization LLM, $\mathbf{s}_j = \textsc{Summarize}(\mathcal{C}_j, \mathcal{P}_j, s_j)$. A fusion LLM then generates a child workflow conditioned on the parent set and summaries: $(\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}}) = \mathcal{F}(\{\mathbf{s}_1, \ldots, \mathbf{s}_J\})$. We evaluate the synthesized workflow on held-out validation data to obtain a scalar reward $r = \mathcal{E}(\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}})$ with $r \in [0, 1]$, insert the new node v_{new} into $\mathcal{G}$ with incoming edges from all parents in a^*, and update the diagonal LinUCB statistics using $\mathbf{x}^*$ (the feature of a^*):

$$A_{kk} \leftarrow A_{kk} + (x_k^*)^2, \qquad b_k \leftarrow b_k + r\, x_k^*, \qquad \theta_k \leftarrow \frac{b_k}{A_{kk}}, \tag{4}$$

for $k = 1, \ldots, Jd$. Finally, we append $(\mathbf{x}^*, r)$ into $\mathcal{B}$ to support occasional refits under the non-stationarity induced by an evolving workflow population. Figure 3 illustrates one optimization iteration: after LinUCB selects a parent tuple, DAGO summarizes each parent's key advantage and then fuses the parents and summaries into a new child workflow, making module inheritance explicit.

Complete Procedure. Algorithm 1 summarizes DAGO. The algorithm runs for at most T iterations (or early-stops after C_{patience} non-improving rounds) and returns the best-scoring workflow found in the DAG. Prompt templates and shared constraints are provided in Appendix D, and implementation-level algorithm details (sampling, diagonal LinUCB, and LLM interaction protocol) are in Appendix E.

Algorithm 1 DAGO: Directed Acyclic Graph Optimization

Require: Task $\mathcal{T}$; iterations T; arm size J; candidates M; init size N; patience C_{patience}
Ensure: Best workflow v^*

```
Initialize workflow DAG G = (V, E)
Generate N initial workflows, evaluate to get s(v), compute embeddings e, insert into V
Initialize LinUCB: A ← λI, b ← 0, θ ← 0; buffer B ← ∅
v* ← arg max_{v∈V} s(v); s* ← s(v*); c ← 0
for t = 1 to T do
    Compute τ_t; sample M arms A_t using p̃^(t)
    Build x_m for all a_m ∈ A_t; select a* via LinUCB
    Summarize parents in a*; fuse to obtain (C_new, P_new) and node v_new
    Evaluate v_new to obtain reward r and embedding e_new
    Insert v_new into G with edges from all parents in a*
    Update (A, b, θ) with (x*, r); add (x*, r) to B
    if r > s* then
        s* ← r; v* ← v_new; c ← 0
    else
        c ← c + 1
    end if
    if c ≥ C_patience then
        break
    end if
end for
return v*
```

Theoretical Justification. DAGO's multi-parent fusion introduces a combinatorial parent-selection problem: at each iteration, the optimizer must choose an ordered J-tuple of parent nodes (an "arm") from an evolving workflow DAG. We view this decision as an *idealized* linear contextual bandit over arm features, where each arm a is represented by the concatenated embedding $\mathbf{x}_a = [\mathbf{e}_1; \ldots; \mathbf{e}_J] \in \mathbb{R}^{Jd}$ and yields a bounded reward $r_t \in [0, 1]$ after LLM fusion and evaluation. DAGO follows a two-stage process: it first *proposes* a candidate set $\mathcal{A}_t$ via annealed score-based sampling (ensuring non-zero coverage), and then applies a LinUCB-style rule to select $a_t \in \mathcal{A}_t$. Under standard linear-bandit conditions (bounded features and conditionally sub-Gaussian noise), LinUCB enjoys sublinear selection regret within the proposed candidates (see Appendix A). In DAGO, we additionally adopt scalable approximations—most notably a diagonalized covariance and a linear surrogate over pretrained embeddings. These deviations can be summarized by a regret-style decomposition that makes the

Table 1. **Main results on six benchmarks.** Test performance (%) averaged over three independent runs. Datasets are grouped by task category (QA / Code / Math), and the macro average across all benchmarks is reported. Best results are shown in **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., 2024b)	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
ADAS (Hu et al., 2025)	64.5	76.6	82.4	53.4	90.8	35.4	67.2
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**

trade-offs explicit:

$$R_T^{\text{sel}} \lesssim \widetilde{\mathcal{O}}\Big(Jd\,L\sqrt{T}\Big) + \sum_{t=1}^{T} \epsilon_t^{\text{diag}} + T\,\epsilon^{\text{lin}}. \quad (5)$$

Here $R_T^{\text{sel}} = \sum_{t=1}^{T} \big(\mu(\mathbf{x}_{a_t^\star}) - \mu(\mathbf{x}_{a_t})\big)$ denotes the regret to the best arm $a_t^\star$ *within* $\mathcal{A}_t$, the first term is the canonical LinUCB rate in dimension Jd, and ϵ_t^{diag} and ϵ^{lin} capture, respectively, the error induced by diagonal uncertainty estimation and by linear-model misspecification along the optimization trajectory. We emphasize that this analysis serves as a *principled rationale* for bandit-guided multi-parent selection, rather than a tight guarantee for the full non-stationary LLM-driven pipeline; full notation and assumptions are provided in Appendix A.

In other words, as DAGO collects fusion feedback, its selection policy becomes increasingly competitive with the best parent-node combination available in $\mathcal{A}_t$ at each round, rather than relying on uninformed random exploration. Concretely, the regret-style guarantee in Equation (5) implies that the average selection suboptimality decreases over time, **so the chosen parent set moves progressively closer to the (candidate-set) optimal combination**, up to the approximation terms.

5. Experiments

5.1. Experimental Setup

Benchmarks. Following AFlow (Zhang et al., 2025d), we evaluate on six public benchmarks across three categories: *Math* (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021)), *Code* (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)), and *QA* (HotpotQA (Yang et al., 2018), DROP (Dua et al., 2019)). We use the same dataset splits and evaluation protocols as AFlow; for MATH we evaluate level-5 problems.

Baselines. We compare DAGO with three families: (i) prompting methods: IO, CoT (Wei et al., 2022), CoT-SC (Wang et al., 2023), MedPrompt (Nori et al., 2023), MultiPersona (Wang et al., 2024b); (ii) iterative refinement: Self-Refine (Madaan et al., 2023), ADAS (Hu et al., 2025); and (iii) automated workflow search: AFlow (Zhang et al., 2025d).

Metrics. We report Solve Rate (%) on GSM8K and MATHlv5*, pass@1 on HumanEval/MBPP (Chen et al., 2021), and F1 on HotpotQA/DROP.

Implementation Details. For all methods, we follow AFlow's implementation and use the same optimization/execution models for controlled comparison. DAGO uses Claude 3.5 Sonnet (Anthropic, 2024) for workflow generation/summarization/fusion and GPT-4o-mini-0718 (OpenAI, 2024) for execution. Each workflow is embedded by Qwen3-Embedding (Zhang et al., 2025e) (d=2560) for bandit-based arm selection. Unless noted otherwise, we set N=10, T=30, M=200, J=5, ν=0.1, and anneal the softmax temperature from τ_{init}=2.0 to τ_{final}=0.5. Full hyperparameters and dataset/reward details are in Appendix B and C.

5.2. Overall Performance and Analysis

Overall Performance. Table 1 and Figure 2 summarize test-set results across all baselines. DAGO achieves the best macro average (81.7%), outperforming AFlow by 1.4 points. As shown in Figure 2(a), the gains are consistent across task categories, including multi-hop QA (+2.4 on HotpotQA), discrete reasoning (+2.1 on DROP), code generation (+1.6 on MBPP), and competition-level mathematics (+1.9 on MATH). On HumanEval, DAGO matches the strongest baseline, indicating that multi-parent fusion preserves high-quality in-domain solutions while improving robustness on the remaining tasks. This overall advantage is also reflected by the method ranking in Figure 2(b), where

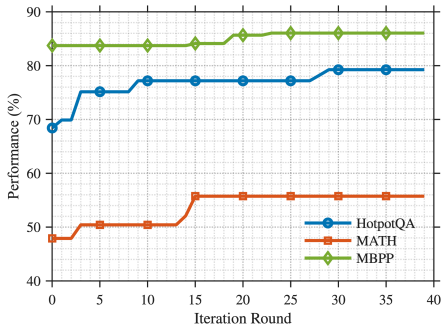


Figure 4. **Validation best-so-far trajectory during optimization.** Running best validation performance over T=30 iterations on HotpotQA, MATH, and MBPP.

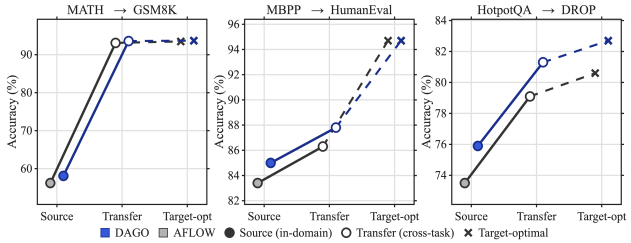


Figure 5. **Cross-task transfer trajectories.** For each transfer pair (source → target), we take the best workflow found on the source task and evaluate it *as-is* on the target task (no re-optimization). Each subplot visualizes: (i) the in-domain performance on the source task, (ii) the transferred performance on the target task, and (iii) the target-task in-domain optimum achieved by the same method under the same optimization budget.

DAGO attains the highest macro-average score.

Optimization Trajectory. Figure 4 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 5 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.

Robust Cost–Performance Trade-offs Across Execution Models. Figure 6 and Table 3 extend the main comparison (which fixes GPT-4o-mini-0718 as the execution model for a controlled and fair evaluation) by assessing robustness across alternative execution models. Across all execution models, DAGO consistently improves upon AFlow in both performance and cost, indicating that the advantage of bandit-guided fusion is not tied to a specific executor. Moreover, stronger execution models can further raise the achievable performance ceiling for both methods, while DAGO maintains a favorable trade-off on the frontier.

Table 2. **Cross-task transfer on the test set (3-run average).** For each transfer pair $s \to t$, we denote by $\mathrm{Opt}_t(m)$ the in-domain best test score achieved by method m when optimizing directly on task t under the same budget, and by $\mathrm{Trans}_{s\to t}(m)$ the test score obtained by transferring the source-optimized best workflow to t without re-optimization. We report transfer regret $R_{s\to t}(m) = \mathrm{Opt}_t(m) - \mathrm{Trans}_{s\to t}(m)$ and its normalized version $\widetilde{R}_{s\to t}(m) = R_{s\to t}(m)/\mathrm{Opt}_t(m)$ (lower is better). All numbers are averaged over three independent runs.

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. **Overall performance vs. total inference cost (test set).** Results for the best workflows from AFlow and DAGO under different execution models. *Overall* denotes the macro average over MATH/MBPP/HotpotQA, and *Total Cost* denotes total inference cost (USD). All numbers are averaged over three independent runs.

Execution Model	Method	Overall↑	Total Cost↓
GPT-4o-mini (OpenAI, 2024)	AFlow	71.0	2.7278
	DAGO	**73.0**	**2.4344**
DeepSeek-V3.2 (Liu et al., 2025)	AFlow	83.7	4.0197
	DAGO	**84.8**	**3.7368**
Gemini-3-Flash (Google, 2025)	AFlow	86.2	14.5263
	DAGO	**87.1**	**10.5712**

5.3. Ablation Study

Effect of Bandit-Guided Arm Selection. Table 4 shows that replacing UCB with random arm selection substantially degrades performance (e.g., −2.9 on GSM8K and −3.1 on HumanEval), indicating that *how to select an arm* is crucial for effective workflow evolution. Moreover, both greedy selection (ν=0) and overly aggressive exploration (ν=1) underperform the balanced default (ν=0.1), supporting the necessity of a principled exploration–exploitation trade-off in navigating the workflow graph.

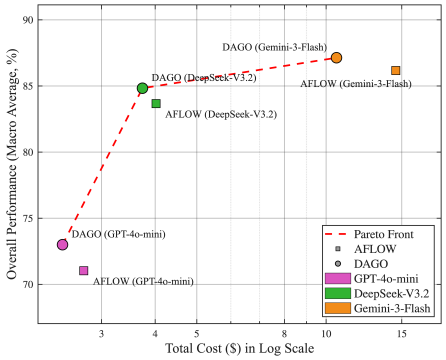


Figure 6. **Performance–cost trade-off under different execution models.** Each point corresponds to the best workflow found by AFlow or DAGO under a given execution model; performance is measured by the macro average over MATH/MBPP/HotpotQA and cost by total inference cost (USD). The dashed line denotes the Pareto frontier.

Table 4. **Ablation on arm selection strategy (3-run test average).** Results on GSM8K, HumanEval, and DROP comparing random arm sampling and UCB-based selection with different exploration strengths ν.

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

Effect of Arm Size. Figure 7 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

This paper challenges the prevailing reliance on random mutation for LLM-based workflow optimization and advocates

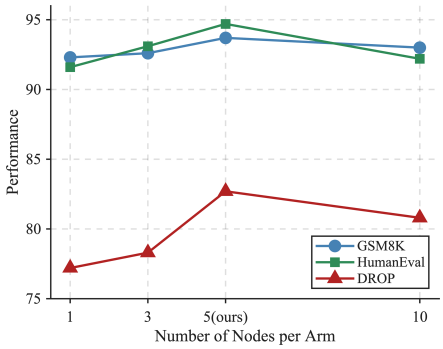


Figure 7. **Effect of arm size J (3-run test average).** Results on GSM8K, HumanEval, and DROP with varying numbers of parents per arm.

Table 5. **Robustness across execution models (3-run test average).** Best workflows from AFlow and DAGO evaluated on MATH, MBPP, and HotpotQA under different execution models.

Execution 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**

a principled shift toward *fusion*. DAGO operationalizes this shift by unifying three key ideas: a DAG-structured memory that enables multi-parent lineage merging, contextual bandit decision-making via Linear UCB for informed parent selection, and LLM-driven fusion to explicitly combine complementary workflow structures. Through extensive experiments across diverse reasoning and coding benchmarks, we show that fusion consistently dominates mutation, bandit-guided selection improves search efficiency over random exploration, and multi-parent derivation yields more robust and transferable workflows than single-parent updates. Together, these components form a coherent optimization framework that not only advances state-of-the-art performance, but also offers a more general and sample-efficient paradigm for automated workflow discovery.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning, specifically the automated discovery and optimization of LLM-based agentic workflows. Our method improves the efficiency and effectiveness of workflow search, which may benefit a wide range of applications in reasoning, code generation, and question answering. We do not anticipate significant negative societal consequences arising directly from this work. As our approach builds upon large language models, it may inherit limitations and

biases present in the underlying models. We therefore encourage practitioners to carefully evaluate generated workflows and apply appropriate safeguards before deployment in real-world or sensitive settings.

References

Abbasi-yadkori, Y., Pál, D., and Szepesvári, C. Improved algorithms for linear stochastic bandits. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. (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.

Ashizawa, R., Hirose, Y., Yoshinari, N., Uchida, K., and Shirakawa, S. Bandit-based prompt design strategy selection improves prompt optimizers. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (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/.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Chen, G., Dong, S., Shu, Y., Zhang, G., Sesay, J., Karlsson, B., Fu, J., and Shi, Y. Autoagents: A framework for automatic agent generation. In Larson, K. (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 22–30. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/3. URL https://doi.org/10.24963/ijcai.2024/3. Main Track.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. 2021.

Chu, W., Li, L., Reyzin, L., and Schapire, R. Contextual bandits with linear payoff functions. In Gordon, G., Dunson, D., and Dudík, M. (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.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Burstein, J., Doran, C., and Solorio, T. (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/.

Gao, H., Liu, Y., He, Y., Dou, L., Du, C., Deng, Z., Hooi, B., Lin, M., and Pang, T. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*, 2025.

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

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. 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.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

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

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., A, S. V., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. 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.

Li, L., Chu, W., Langford, J., and Schapire, R. E. 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.

Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Ma, Z., Zhao, Z., Hua, C., Berto, F., and Park, J. Judgeflow: Agentic workflow optimization via block judge. *arXiv preprint arXiv:2601.07477*, 2026.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Mens, T. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 28(5):449–462, May 2002. ISSN 1939-3520. doi: 10.1109/TSE.2002.1000449.

Mitchell, M. *An introduction to genetic algorithms*. MIT press, 1998.

Niu, B., Song, Y., Lian, K., Shen, Y., Yao, Y., Zhang, K., and Liu, T. Flow: Modularized agentic workflow automation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sLKDbuyq99.

Nori, H., Lee, Y. T., Zhang, S., Carignan, D., Edgar, R., Fusi, N., King, N., Larson, J., Li, Y., Liu, W., 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.

Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., and Khattab, O. Optimizing instructions and demonstrations for multi-stage language model programs. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (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/.

Panda, P., Magazine, R., Devaguptapu, C., Takemori, S., and Sharma, V. Adaptive LLM routing under budget constraints. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (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/.

Shi, C., Yang, K., Chen, Z., Li, J., Yang, J., and Shen, C. 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.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Tang, X., Gao, Q., Li, J., Du, N., Li, Q., and Xie, S. MBA-RAG: a bandit approach for adaptive retrieval-augmented generation through question complexity. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B. D., and Schockaert, S. (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/.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. A survey on

large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024a.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. 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.

Wang, Y., Yang, L., Li, G., Wang, M., and Aragam, B. Scoreflow: Mastering llm agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*, 2025.

Wang, Z., Mao, S., Wu, W., Ge, T., Wei, F., and Ji, H. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In Duh, K., Gomez, H., and Bethard, S. (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 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.15. URL https://aclanthology.org/2024.naacl-long.15/.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., brian ichter, Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Xie, M., Chen, S., and Lv, C. A component-based survey of interactions between large language models and multi-armed bandits. *arXiv preprint arXiv:2601.12945*, 2026.

Xu, S., Zhang, J., Di, S., Luo, Y., Yao, L., Liu, H., Zhu, J., Liu, F., and Zhang, M.-L. Robustflow: Towards robust agentic workflow generation. *arXiv preprint arXiv:2509.21834*, 2025.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (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/.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Lu, P., Huang, Z., Guestrin, C., and Zou, J. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Zhang, G., Chen, K., Wan, G., Chang, H., Cheng, H., Wang, K., Hu, S., and Bai, L. Evoflow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*, 2025a.

Zhang, G., Niu, L., Fang, J., Wang, K., BAI, L., and Wang, X. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=imcyVlzpXh.

Zhang, G., Yue, Y., Sun, X., Wan, G., Yu, M., Fang, J., Wang, K., Chen, T., and Cheng, D. 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.

Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X.-H., Chen, J., Zhuge, M., Cheng, X., Hong, S., Wang, J., Zheng, B., Liu, B., Luo, Y., and Wu, C. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025d. URL https://openreview.net/forum?id=z5uVAKwmjf.

Zhang, Y., Li, M., Long, D., Zhang, X., Lin, H., Yang, B., Xie, P., Yang, A., Liu, D., Lin, J., et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025e.

Zheng, C., Chen, J., Lyu, Y., Ng, W. Z. T., Zhang, H., Ong, Y.-S., Tsang, I., and Yin, H. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming. *arXiv preprint arXiv:2505.22967*, 2025.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Zhuge, M., Wang, W., Kirsch, L., Faccio, F., Khizbullin, D., and Schmidhuber, J. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

Appendix

A. Theoretical Perspective on Bandit-Guided Multi-Parent Fusion

This section provides an idealized analysis that clarifies *what* DAGO's LinUCB component is optimizing and *which* assumptions are sufficient for standard guarantees. The discussion is meant as a principled rationale aligned with Sections 3 and 4, rather than a claim of a tight regret guarantee for the full LLM-driven pipeline.

Setup and notation. At iteration t, DAGO maintains a workflow DAG $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. A *fusion arm* is an ordered J-tuple of parent nodes $a = (v_1, \ldots, v_J)$, represented by the concatenated embedding $\mathbf{x}_a = [\mathbf{e}_1; \ldots; \mathbf{e}_J] \in \mathbb{R}^p$ where $p = Jd$ (Section 4). DAGO constructs a candidate set $\mathcal{A}_t$ by annealed score sampling, then selects

$$a_t \in \arg\max_{a \in \mathcal{A}_t} \mathbf{x}_a^\top \hat{\boldsymbol{\theta}}_{t-1} \; + \; \nu \sqrt{\mathbf{x}_a^\top \widehat{\mathbf{A}}_{t-1}^{-1} \mathbf{x}_a}, \tag{6}$$

where $\widehat{\mathbf{A}}_{t-1}$ is the (regularized) empirical covariance of previously selected arms; in DAGO we use its diagonal approximation for $O(p)$ updates (Section 4). After LLM summarization and fusion, the synthesized child workflow is evaluated to obtain a bounded reward $r_t \in [0, 1]$ (Section 4).

A minimal set of assumptions. We state a standard linear contextual bandit model as a *surrogate* for predicting the fusion reward from arm features.

Assumption A.1 (Bounded features and conditionally sub-Gaussian noise)**.** For all t and all $a \in \mathcal{A}_t$, $\|\mathbf{x}_a\|_2 \le L$. Rewards are generated as $r_t = \mu(\mathbf{x}_{a_t}) + \eta_t$ with $\mu(\mathbf{x}) \in [0, 1]$ and $\{\eta_t\}$ forming a martingale difference sequence w.r.t. the filtration $\{\mathcal{F}_t\}$: $\mathbb{E}[\eta_t \mid \mathcal{F}_{t-1}] = 0$ and η_t is σ-sub-Gaussian conditionally on $\mathcal{F}_{t-1}$.

Assumption A.2 (Realizable linear model (reference setting))**.** There exists $\boldsymbol{\theta}^\star \in \mathbb{R}^p$ with $\|\boldsymbol{\theta}^\star\|_2 \le S$ such that $\mu(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta}^\star$ for all arm features encountered during optimization.

Assumption A.2 should be interpreted as an *idealization* that isolates the role of LinUCB in DAGO: the true mapping from parent workflows to post-fusion performance is mediated by an LLM and can be highly non-linear. Nevertheless, this surrogate model is useful because DAGO's arm features are produced by a pretrained semantic encoder, and LinUCB only needs to be predictive *locally* along the trajectory of encountered arms to guide exploration.

Candidate sets may vary across rounds. Unlike textbook contextual bandits with a fixed action set, DAGO constructs $\mathcal{A}_t$ by sampling from the current DAG (Section 4). Importantly, standard LinUCB analyses do *not* require a fixed action set; they hold for any (possibly time-varying) sequence $\{\mathcal{A}_t\}_{t=1}^T$ as long as features are bounded and measurable w.r.t. $\mathcal{F}_{t-1}$, since the proof is driven by the self-normalized concentration of the realized feature sequence. We therefore define regret *within* the proposed candidate sets.

Definition A.3 (Candidate-optimal arm and selection regret)**.** Let $a_t^\star \in \arg\max_{a \in \mathcal{A}_t} \mu(\mathbf{x}_a)$ be the best arm in the candidate set. The selection regret is

$$R_T^{\mathrm{sel}} \; = \; \sum_{t=1}^{T} \big(\mu(\mathbf{x}_{a_t^\star}) - \mu(\mathbf{x}_{a_t})\big). \tag{7}$$

Proposition A.4 (Reference regret bound for LinUCB over proposed candidates)**.** *Under Assumptions A.1–A.2, if ν in Equation (6) is chosen as in standard LinUCB (e.g., the confidence radius derived from ridge regression), then with probability at least $1 - \delta$,*

$$R_T^{\mathrm{sel}} \; \le \; \widetilde{\mathcal{O}}\Big(p\, L\, \sqrt{T}\Big) \; = \; \widetilde{\mathcal{O}}\Big(J d\, L\, \sqrt{T}\Big), \tag{8}$$

where $\widetilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors in $(T, 1/\delta, \lambda)$.

This proposition is a direct specialization of standard linear contextual bandit regret bounds for LinUCB / OFUL-style algorithms (e.g., Li et al. (2010); Chu et al. (2011); Abbasi-yadkori et al. (2011)) to the concatenated feature dimension $p = Jd$.

Two-stage view: proposal vs. selection. DAGO is a two-stage decision process: (i) a *proposal policy* over the evolving DAG produces $\mathcal{A}_t$ (annealed score sampling with a nonzero floor $p_{\min}$), and (ii) LinUCB selects a_t from $\mathcal{A}_t$. To make this

explicit, define an (ideal) full combinatorial action set $\bar{\mathcal{A}}_t \subseteq \mathcal{V}_t^J$ that contains all ordered J-tuples of current nodes, and let $\bar{a}_t^\star \in \arg\max_{a \in \bar{\mathcal{A}}_t} \mu(\mathbf{x}_a)$. Then the total regret to the globally best fusion at each round decomposes as

$$\sum_{t=1}^{T} \big(\mu(\mathbf{x}_{\bar{a}_t^\star}) - \mu(\mathbf{x}_{a_t})\big) \;=\; \underbrace{\sum_{t=1}^{T} \big(\mu(\mathbf{x}_{\bar{a}_t^\star}) - \mu(\mathbf{x}_{a_t^\star})\big)}_{\text{proposal gap}} \;+\; \underbrace{R_T^{\text{sel}}}_{\text{selection regret}} . \tag{9}$$

The second term is controlled by LinUCB as in Equation (8). The first term captures the quality of the candidate-arm proposal mechanism: annealed score sampling concentrates probability mass on promising parents while maintaining coverage through $p_{\min}$ (Section 4). In particular, the nonzero floor implies that every node remains selectable as a parent with nonzero probability at every round, which prevents the proposal distribution from collapsing to a strict subset of $\mathcal{V}_t$.

Accounting for approximation and misspecification (DAGO setting). DAGO uses two additional approximations motivated by scalability:

(i) Diagonal uncertainty. Let $\mathbf{A}_t$ denote the full covariance and $\mathbf{D}_t = \mathrm{diag}(\mathbf{A}_t)$ its diagonal. DAGO replaces $\sqrt{\mathbf{x}^\top \mathbf{A}_t^{-1} \mathbf{x}}$ by $\sqrt{\mathbf{x}^\top \mathbf{D}_t^{-1} \mathbf{x}}$ to obtain $O(p)$ time/space per update (Section 4). Define the per-round uncertainty approximation error

$$\epsilon_t^{\text{diag}} \;=\; \sup_{\mathbf{x} \in \mathcal{X}_t} \left| \sqrt{\mathbf{x}^\top \mathbf{A}_t^{-1} \mathbf{x}} - \sqrt{\mathbf{x}^\top \mathbf{D}_t^{-1} \mathbf{x}} \right|, \tag{10}$$

where $\mathcal{X}_t = \{\mathbf{x}_a : a \in \mathcal{A}_t\}$ is the encountered feature set at round t. This quantity is problem-dependent and reflects the strength of feature correlations that are ignored by the diagonal metric.

(ii) Model misspecification. More generally, the expected fusion reward may not be exactly linear in the embeddings. A standard way to express this is

$$\mu(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta}^\star + \Delta(\mathbf{x}), \qquad \text{with} \quad |\Delta(\mathbf{x})| \le \epsilon^{\text{lin}} \;\text{ for all encountered } \mathbf{x}, \tag{11}$$

where ϵ^{lin} measures the deviation from the realizable linear surrogate along the optimization trajectory.

Combining Equations (10) and (11) with Equation (8) yields the following *regret-style* interpretation: the selection performance can be viewed as the reference LinUCB term plus additive contributions that scale with the cumulative approximation along the trajectory,

$$R_T^{\text{sel}} \;\lesssim\; \widetilde{\mathcal{O}}\Big(Jd\,L\sqrt{T}\Big) \;+\; \sum_{t=1}^{T} \epsilon_t^{\text{diag}} \;+\; T\,\epsilon^{\text{lin}}. \tag{12}$$

We emphasize that Equation (12) is intended as a *qualitative* diagnostic: it makes explicit how (a) increasing arm size J enlarges the feature dimension, and (b) diagonal uncertainty and linear surrogacy influence bandit guidance, while keeping the algorithm computationally feasible in the high-dimensional concatenated embedding space.

Interpretation for finite-budget workflow optimization. In DAGO, each iteration requires an expensive LLM fusion and workflow evaluation (Section 4), so the optimization budget T is small relative to classical bandit regimes. The analysis above therefore serves to justify the algorithmic design choices that prioritize *sample-efficient exploration* (UCB guidance within candidates) and *controlled coverage* (annealed proposal with a nonzero floor), rather than to optimize asymptotic constants. Empirically, this combination is sufficient to consistently identify high-performing multi-parent fusions under strict evaluation budgets across diverse benchmarks (see Section 5).

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 6 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 6. 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.
	Early-stopping patience C_{patience}	5	Stop if the best validation score does not improve for C_{patience} consecutive iterations.
	Minimum iterations $T_{\min}$	20	Enable early stopping only after at least $T_{\min}$ iterations.
Arm proposal	Candidate arms M	200	Number of candidate parent tuples proposed per iteration.
	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.1).
	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; if $\lvert\mathcal{V}\rvert < J$, we set $J \leftarrow \lvert\mathcal{V}\rvert$.
LinUCB	Exploration coefficient ν	0.1	Exploration weight in Eq. (1); ν is kept constant by default.
	Regularization λ	0.2	Ridge regularization in the bandit covariance (Eq. (1)).
	Covariance approximation	Diagonal	Maintain only $\mathrm{diag}(\mathbf{A})$ to compute uncertainty in $O(Jd)$ (Appendix E.2).
	Reward r	$[0, 1]$	Child validation score, normalized to $[0, 1]$, used as the bandit reward.
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.2).
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	20–50	Number of validation instances used to estimate rewards during search (dataset-dependent; fixed across iterations).
	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 7. 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. Final evaluation is performed on the full held-out test subset in Table 7.

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. Extended Algorithm Details

This appendix provides implementation-level details that complement Section 4. We reuse the notation defined in Section 3: at iteration t, DAGO maintains a workflow DAG $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. Each node $v \in \mathcal{V}_t$ corresponds to an executable workflow with validation score $s(v) \in [0, 1]$ and a fixed semantic embedding $\mathbf{e}(v) \in \mathbb{R}^d$. An *arm* (candidate fusion configuration) is an ordered J-tuple of parent nodes $a = (v_1, \ldots, v_J)$, and its contextual feature is constructed by concatenation $\mathbf{x}(a) = [\mathbf{e}(v_1); \ldots; \mathbf{e}(v_J)] \in \mathbb{R}^p$ with $p = Jd$ (Sections 3 and 4).

E.1. Softmax Temperature Sampling

DAGO proposes a candidate arm set $\mathcal{A}_t$ by temperature-annealed softmax sampling over the current node pool. The goal is to (i) allocate more probability mass to high-performing workflows, while (ii) guaranteeing nonzero coverage to prevent premature collapse.

Score-to-probability mapping. Let $\mathcal{V}_t$ be the current node set with $n_t = |\mathcal{V}_t|$ and scores $\{s(v)\}_{v \in \mathcal{V}_t}$. Given a temperature $\tau_t > 0$, we define unnormalized weights

$$w_t(v) \;=\; \exp\!\left(\frac{s(v)}{\tau_t}\right), \qquad v \in \mathcal{V}_t, \tag{13}$$

and the corresponding softmax distribution

$$p_t(v) \;=\; \frac{w_t(v)}{\sum_{u\in\mathcal{V}_t} w_t(u)}. \tag{14}$$

(In practice, for numerical stability one may compute $w_t(v) = \exp((s(v) - s_{\max})/\tau_t)$ with $s_{\max} = \max_{u\in\mathcal{V}_t} s(u)$; this does not change p_t.)

Exploration floor and renormalization. To ensure persistent exploration, we apply a probability floor $p_{\min} > 0$ and renormalize:

$$\hat{p}_t(v) \;=\; \max\big(p_t(v),\, p_{\min}\big), \qquad \tilde{p}_t(v) \;=\; \frac{\hat{p}_t(v)}{\sum_{u\in\mathcal{V}_t} \hat{p}_t(u)}. \tag{15}$$

The floor guarantees $\tilde{p}_t(v) \geq \frac{p_{\min}}{\sum_u \hat{p}_t(u)} > 0$ for all nodes, so every workflow remains selectable as a parent throughout the run.

Temperature annealing. We anneal temperature linearly over the total budget T:

$$\tau_t \;=\; \tau_{\text{init}} + \frac{t}{T}\big(\tau_{\text{final}} - \tau_{\text{init}}\big), \qquad t = 1, \ldots, T, \tag{16}$$

where typically $\tau_{\text{init}} > \tau_{\text{final}}$ to encourage broad exploration early and more exploitative sampling later.

Candidate-arm construction. To build a candidate arm set $\mathcal{A}_t$ of size M, we repeatedly sample J *distinct* parents *without replacement* from $\mathcal{V}_t$ according to $\tilde{\mathbf{p}}_t$ and form an arm $a = (v_{m_1}, \ldots, v_{m_J})$. If $|\mathcal{V}_t| < J$, we set $J \leftarrow |\mathcal{V}_t|$ for that round. To impose consistent positional semantics for concatenation, we order the sampled parents by descending validation score (ties may be broken deterministically, e.g., by node id). Optionally, we de-duplicate candidate arms by treating arms as *sets* for duplicate checking (i.e., ignoring within-arm order only for uniqueness tests), which increases diversity in $\mathcal{A}_t$.

Algorithm 2 Candidate-arm proposal by annealed score sampling

Require: Workflow nodes $\mathcal{V}_t$ with scores $s(\cdot)$; candidates M; parents per arm J; probability floor $p_{\min}$; temperatures $(\tau_{\text{init}}, \tau_{\text{final}})$; total budget T
Ensure: Candidate arm set $\mathcal{A}_t$
$\tau_t \leftarrow \tau_{\text{init}} + (\tau_{\text{final}} - \tau_{\text{init}}) \cdot \frac{t}{T}$
$J \leftarrow \min(J, |\mathcal{V}_t|)$
$w_t(v) \leftarrow \exp(s(v)/\tau_t),\;\; p_t(v) \leftarrow w_t(v)/\sum_{u\in\mathcal{V}_t} w_t(u)$
$\hat{p}_t(v) \leftarrow \max(p_t(v), p_{\min});\;\; \tilde{p}_t(v) \leftarrow \hat{p}_t(v)/\sum_{u\in\mathcal{V}_t} \hat{p}_t(u)$
$\mathcal{A}_t \leftarrow \emptyset$
while $|\mathcal{A}_t| < M$ **do**
  Sample J distinct parents $\{v_1, \ldots, v_J\} \sim \tilde{\mathbf{p}}_t$ without replacement
  Order by score: $a \leftarrow (v_{(1)}, \ldots, v_{(J)})$ s.t. $s(v_{(1)}) \geq \cdots \geq s(v_{(J)})$
  If a is new (optional de-duplication), add a to $\mathcal{A}_t$
end while
return $\mathcal{A}_t$

E.2. Linear UCB with Diagonal Approximation

DAGO formulates arm selection as a contextual linear bandit over the concatenated feature space. Let $p = Jd$ and assume the (surrogate) linear reward model

$$\mathbb{E}[r_t \mid \mathbf{x}_t] \;=\; \mathbf{x}_t^\top \boldsymbol{\theta}, \qquad \boldsymbol{\theta} \in \mathbb{R}^p, \tag{17}$$

where $r_t \in [0, 1]$ is the observed reward at iteration t (the child workflow's validation score; Section 4). Standard LinUCB maintains a regularized covariance $\mathbf{A} \in \mathbb{R}^{p\times p}$ and $\mathbf{b} \in \mathbb{R}^p$, producing $\hat{\boldsymbol{\theta}} = \mathbf{A}^{-1}\mathbf{b}$ and selecting the arm that maximizes an upper confidence bound. Since $p = Jd$ is large in our setting, DAGO uses a *diagonal* approximation.

Diagonal state. We store only the diagonal of the regularized covariance:

$$\mathbf{A}_0 \;=\; \lambda \mathbf{I}_p \quad\Longrightarrow\quad \mathbf{a}_0 \;=\; \lambda \mathbf{1}_p, \qquad \mathbf{b}_0 \;=\; \mathbf{0}_p, \tag{18}$$

where $\mathbf{a}_t = \mathrm{diag}(\mathbf{A}_t) \in \mathbb{R}^p$ and $\lambda > 0$ is ridge regularization. The diagonal ridge estimate is computed element-wise:

$$\hat{\boldsymbol{\theta}}_t \;=\; \mathbf{b}_t \oslash \mathbf{a}_t, \tag{19}$$

where $\oslash$ denotes element-wise division.

UCB scoring under diagonal uncertainty. Given a candidate arm feature $\mathbf{x} \in \mathbb{R}^p$, we compute

$$\mu(\mathbf{x}) \;=\; \mathbf{x}^\top \hat{\boldsymbol{\theta}}_{t-1}, \qquad \sigma(\mathbf{x}) \;=\; \nu \sqrt{\mathbf{x}^\top \mathbf{A}_{t-1}^{-1} \mathbf{x}} \;\approx\; \nu \sqrt{\sum_{k=1}^{p} \frac{x_k^2}{a_{t-1,k}}}, \tag{20}$$

and select

$$a_t \in \arg\max_{a \in \mathcal{A}_t} \Big(\mu(\mathbf{x}(a)) + \sigma(\mathbf{x}(a)) \Big), \tag{21}$$

where $\nu > 0$ controls exploration strength.

Online update. After executing summarization and fusion on the selected arm a_t to obtain a child workflow, we evaluate it on the validation split to get reward $r_t \in [0, 1]$, and update the diagonal statistics:

$$\mathbf{a}_t \;\leftarrow\; \mathbf{a}_{t-1} + \mathbf{x}_t \odot \mathbf{x}_t, \qquad \mathbf{b}_t \;\leftarrow\; \mathbf{b}_{t-1} + r_t \, \mathbf{x}_t, \tag{22}$$

where $\odot$ is element-wise multiplication and $\mathbf{x}_t = \mathbf{x}(a_t)$.

Algorithm 3 Diagonal Linear UCB (LinUCB) for arm selection and update

Require: Embedding dimension d; arm size J; feature dim $p = Jd$; exploration ν; regularization λ
Initialize: $\mathbf{a} \leftarrow \lambda \cdot \mathbf{1}_p$, $\mathbf{b} \leftarrow \mathbf{0}_p$
for each iteration t **do**
  Construct candidate set $\mathcal{A}_t$ (Appendix E.1)
  $\hat{\boldsymbol{\theta}} \leftarrow \mathbf{b} \oslash \mathbf{a}$ {Diagonal ridge estimate}
  for each arm $a \in \mathcal{A}_t$ with feature $\mathbf{x} = \mathbf{x}(a) \in \mathbb{R}^p$ **do**
    $\mu(a) \leftarrow \mathbf{x}^\top \hat{\boldsymbol{\theta}}$
    $\sigma(a) \leftarrow \nu \sqrt{\sum_{k=1}^{p} x_k^2 / a_k}$
    $\mathrm{UCB}(a) \leftarrow \mu(a) + \sigma(a)$
  end for
  Select $a_t \leftarrow \arg\max_{a \in \mathcal{A}_t} \mathrm{UCB}(a)$ and obtain reward $r_t \in [0, 1]$ after fusion+evaluation
  $\mathbf{x}_t \leftarrow \mathbf{x}(a_t)$
  $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{x}_t \odot \mathbf{x}_t$ {Update diag covariance}
  $\mathbf{b} \leftarrow \mathbf{b} + r_t \cdot \mathbf{x}_t$ {Update reward-weighted features}
end for

Complexity. The diagonal approximation reduces covariance storage from $O(p^2)$ to $O(p)$ and avoids matrix inversion. Per-iteration scoring over M candidates costs $O(Mp)$ (dominated by dot products and diagonal uncertainty), and the online update costs $O(p)$. This makes LinUCB guidance practical under high-dimensional concatenated embeddings ($p = Jd$) and small evaluation budgets.

E.3. Embedding and Arm Feature Construction

Node embedding. Each workflow node v_i is associated with structural code $\mathcal{C}_i$ and an internal prompt set $\mathcal{P}_i$. We obtain a fixed embedding by encoding their textual concatenation:

$$\mathbf{e}_i \;=\; \mathrm{Enc}(\mathcal{C}_i \oplus \mathcal{P}_i) \in \mathbb{R}^d, \tag{23}$$

where $\oplus$ denotes string concatenation and $\mathrm{Enc}(\cdot)$ is a pretrained embedding model. Since $(\mathcal{C}_i, \mathcal{P}_i)$ is immutable once inserted into the DAG, embeddings are computed once, cached, and reused across all subsequent iterations.

Arm feature (concatenation). Given a candidate arm $a_m = (v_{m_1}, \ldots, v_{m_J})$ whose parents are ordered by descending score (Appendix E.1), we form the LinUCB feature

$$\mathbf{x}_m \;=\; [\mathbf{e}_{m_1}; \ldots; \mathbf{e}_{m_J}] \in \mathbb{R}^{Jd}. \tag{24}$$

This design preserves *positional* information: the first slot typically corresponds to the strongest parent, allowing the linear model to learn asymmetric contributions across parent roles (e.g., a "base" parent vs. auxiliary parents). (Alternative commutative aggregations, such as summation, remove such positional semantics but were not adopted in DAGO.)

E.4. LLM Interaction Protocol (Initial / Summarize / Fuse)

The LLM component is structured to minimize format errors and uncontrolled drift, while still enabling semantic recombination.

(i) Strategy discovery and initialization. DAGO first performs task-specific strategy discovery to obtain diverse high-level workflow blueprints. It then generates an initial population of N workflows consistent with the discovered strategies, evaluates them to obtain $s(v)$, computes embeddings, and inserts them into $\mathcal{G}_0$.

(ii) Parent advantage summarization. Before fusion at iteration t, DAGO applies a dedicated summarization model to each selected parent $v_j = (\mathcal{C}_j, \mathcal{P}_j)$ to extract concise, evaluator-aware guidance:

$$\mathbf{z}_j \;=\; \mathrm{Summarize}(\mathcal{C}_j,\, \mathcal{P}_j,\, s(v_j))\,, \tag{25}$$

where $\mathbf{z}_j$ is a structured summary (e.g., key strengths, failure modes, and alignment risks) used as auxiliary evidence for fusion.

(iii) Multi-parent fusion with constrained edits. Given the ordered parent set $a_t = (v_1, \ldots, v_J)$ and summaries $\{\mathbf{z}_j\}_{j=1}^{J}$, a fusion model generates a child workflow by starting from the best parent and applying a small number of justified, targeted modifications:

$$(\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}}) \;=\; \mathrm{Fuse}\big(\{(\mathcal{C}_j, \mathcal{P}_j, s(v_j), \mathbf{z}_j)\}_{j=1}^{J}\big)\,. \tag{26}$$

The synthesized child is evaluated on the held-out validation split to produce $r_t = s(v_{\text{new}}) \in [0, 1]$, which is then used to update the bandit statistics (Appendix E.2). All optimizer-facing prompts follow a strict, machine-parseable output schema so that $\mathcal{C}_{\text{new}}$ and $\mathcal{P}_{\text{new}}$ can be extracted deterministically; the concrete templates are listed in Appendix D.

E.5. End-to-end DAGO Optimization Loop

Algorithm 4 summarizes the complete DAGO optimization loop. Starting from an initial population, we maintain a workflow DAG $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ where each node stores executable workflow code/prompts $(\mathcal{C}(v), \mathcal{P}(v))$, its validation score $s(v) \in [0, 1]$, and a cached embedding $\mathbf{e}(v) = \mathrm{Enc}(\mathcal{C}(v) \oplus \mathcal{P}(v))$. At each iteration t, DAGO (i) proposes M candidate parent tuples (arms) by annealed softmax sampling over $\{s(v)\}$ with a probability floor $p_{\min}$, (ii) scores each arm using diagonal LinUCB on the concatenated feature $\mathbf{x}(a) = [\mathbf{e}(v_1); \ldots; \mathbf{e}(v_{J_t})]$ and selects the maximizer, (iii) summarizes each selected parent via $\mathrm{SummarizeLLM}$ and fuses them into a child workflow via $\mathrm{FuseLLM}$, and (iv) evaluates the child to obtain reward $r_t = \mathrm{Eval}(\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}})$, inserts it into the DAG with edges from all parents, and updates the bandit statistics with $(\mathbf{x}_t, r_t)$. This design couples score-guided exploration with uncertainty-aware arm selection, enabling systematic multi-parent fusion while preserving acyclicity by always linking existing nodes to newly created descendants.

Algorithm 4 DAGO: Bandit-guided multi-parent fusion over a workflow DAG (end-to-end)

Require: Dataset $\mathcal{D}$ (validation split) and evaluator $\mathrm{Eval}(\cdot) \in [0,1]$; encoder $\mathrm{Enc}(\cdot) \to \mathbb{R}^d$; LLMs $\{\mathrm{StrategyLLM}, \mathrm{InitLLM}, \mathrm{SummarizeLLM}, \mathrm{FuseLLM}\}$; budgets N, T, M, J; sampling $p_{\min}$, $(\tau_{\text{init}}, \tau_{\text{final}})$; LinUCB λ, ν.

Ensure: Best node $v^\star$ and $\mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T)$.

// (0) Strategy discovery
$\mathcal{S} \leftarrow \mathrm{StrategyLLM}(\mathcal{D})$
// (1) Initialization: generate/evaluate initial population
$\mathcal{V}_0 \leftarrow \emptyset,\ \mathcal{E}_0 \leftarrow \emptyset$; cache $\mathcal{H} \leftarrow \emptyset$
for $k = 1$ to N **do**
  $(\mathcal{C}_k, \mathcal{P}_k) \leftarrow \mathrm{InitLLM}(\mathcal{D}, \mathcal{S})$
  $s_k \leftarrow \mathrm{Eval}(\mathcal{C}_k, \mathcal{P}_k)$; $t_k \leftarrow \mathcal{C}_k \oplus \mathcal{P}_k$
  if $t_k \notin \mathcal{H}$ **then** $\mathcal{H}[t_k] \leftarrow \mathrm{Enc}(t_k)$
  Add $v_k \triangleq (\mathcal{C}_k, \mathcal{P}_k, s_k, \mathcal{H}[t_k])$ to $\mathcal{V}_0$
end for
$v^\star \leftarrow \arg\max_{v \in \mathcal{V}_0} s(v)$
// (2) Initialize diagonal LinUCB in feature dim $p = Jd$
$p \leftarrow Jd$; $\mathbf{a} \leftarrow \lambda \mathbf{1}_p$; $\mathbf{b} \leftarrow \mathbf{0}_p$
// (3) Main optimization loop
for $t = 1$ to T **do**
  $J_t \leftarrow \min(J, |\mathcal{V}_{t-1}|)$; $\tau_t \leftarrow \tau_{\text{init}} + (\tau_{\text{final}} - \tau_{\text{init}})\frac{t}{T}$
  // (3.1) Candidate arms via annealed softmax sampling
  Compute $\tilde{p}_t(\cdot)$ from scores $\{s(v)\}_{v \in \mathcal{V}_{t-1}}$ using τ_t and floor $p_{\min}$ (App. E.1)
  Propose $\mathcal{A}_t$ of size M by sampling J_t distinct parents from $\tilde{p}_t$ and ordering each arm by descending score
  // (3.2) Diagonal LinUCB scoring and selection
  $\hat{\boldsymbol{\theta}} \leftarrow \mathbf{b} \oslash \mathbf{a}$
  for each $a_m = (v_{m_1}, \ldots, v_{m_{J_t}}) \in \mathcal{A}_t$ **do**
    $\mathbf{x}_m \leftarrow [\mathbf{e}(v_{m_1}); \ldots; \mathbf{e}(v_{m_{J_t}})]$
    $\mathrm{UCB}_m \leftarrow \mathbf{x}_m^\top \hat{\boldsymbol{\theta}} + \nu \sqrt{\sum_k x_{m,k}^2 / a_k}$
  end for
  $m^\star \leftarrow \arg\max_m \mathrm{UCB}_m$; $a_t \leftarrow a_{m^\star}$; $\mathbf{x}_t \leftarrow \mathbf{x}_{m^\star}$
  // (3.3) Parent summarization
  for each $v_j \in a_t$ **do**
    $\mathbf{z}_j \leftarrow \mathrm{SummarizeLLM}(\mathcal{C}(v_j), \mathcal{P}(v_j), s(v_j))$
  end for
  // (3.4) Multi-parent fusion
  $(\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}}) \leftarrow \mathrm{FuseLLM}\big(\{(\mathcal{C}(v_j), \mathcal{P}(v_j), s(v_j), \mathbf{z}_j)\}_{v_j \in a_t}\big)$
  // (3.5) Evaluate, embed, and insert child
  $r_t \leftarrow \mathrm{Eval}(\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}})$; $t_{\text{new}} \leftarrow \mathcal{C}_{\text{new}} \oplus \mathcal{P}_{\text{new}}$
  if $t_{\text{new}} \notin \mathcal{H}$ **then** $\mathcal{H}[t_{\text{new}}] \leftarrow \mathrm{Enc}(t_{\text{new}})$
  Create $v_{\text{new}} \triangleq (\mathcal{C}_{\text{new}}, \mathcal{P}_{\text{new}}, r_t, \mathcal{H}[t_{\text{new}}])$; add v_{new} to $\mathcal{V}_t$
  Add edges $(v_j \to v_{\text{new}})$ for all $v_j \in a_t$ to $\mathcal{E}_t$; **if** $r_t > s(v^\star)$ then $v^\star \leftarrow v_{\text{new}}$
  // (3.6) Diagonal LinUCB update
  $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{x}_t \odot \mathbf{x}_t$; $\mathbf{b} \leftarrow \mathbf{b} + r_t\, \mathbf{x}_t$
end for
return $v^\star,\ \mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T)$

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, all scores reported in this appendix correspond to the *validation reward used during optimization* (averaged over three independent executions), rather than the final test-set results in Table 1.

F.1. Cross-Dataset Optimization Statistics

Table 8 summarizes the optimization trajectories across all benchmarks, revealing several important patterns about DAGO's search dynamics.

Table 8. Detailed DAGO optimization statistics per benchmark, including convergence analysis and multi-parent fusion characteristics. Scores are validation rewards used during optimization (averaged over three independent 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%	84.9%
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%	-1.1%
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 8:

(1) Task-Dependent Convergence Behavior. Mathematical reasoning (GSM8K, MATH) and code generation (HumanEval, MBPP) tasks triggered early stopping, indicating convergence to stable local optima within 25–28 iterations. In contrast, question answering tasks (HotpotQA, DROP) ran the full 30 iterations without early stopping, suggesting a more complex optimization landscape with continued exploration opportunities. This aligns with the higher fusion-reward variance observed in these tasks (12.0% and 6.0% respectively).

(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 using Programmer operator
        verify = await self.programmer(
            problem=f"Verify: {initial['response']}",
            analysis="Extract and verify numerical calculations")

        # Step 3: Generate 2 alternative solutions
        alts = []
        for _ in range(2):
            alt = await self.custom(input=problem,
                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 explicitly verifies the initial solution before generating alternatives, creating a feedback loop. AFlow generates all solutions independently without cross-verification.
2. **Solution Coupling**: DAGO's alternatives are generated *after* seeing the initial solution and verification result, enabling informed diversity. AFlow's solutions are generated in parallel without information sharing.
3. **Computational Efficiency**: DAGO uses 5 LLM calls (1 solve + 1 verify + 2 alternatives + 1 ensemble) vs. AFlow's 6 calls (1 programmer + 4 custom + 1 ensemble), achieving better performance with fewer calls.

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)

```
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)

```
# 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)

```
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.

Interesting Observation: Initial vs. Fusion Performance. DROP is unique in that the best initial node (Node 9, 84.9%) slightly outperforms the best fusion node (Node 25, 84.0%). This suggests that for some tasks, the initial diverse population may already contain near-optimal solutions, and fusion primarily serves to maintain robust performance rather than discover strictly superior workflows. Importantly, DAGO does not degrade performance through fusion—the optimization process identifies and preserves high-performing patterns.

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 9. 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 10. 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 out of six datasets triggered early stopping due to convergence, while HotpotQA and DROP required the full 30 iterations.

G.2. Detailed Optimization Statistics

Table 11 provides comprehensive statistics capturing the optimization dynamics for each benchmark.

Table 11. DAGO optimization statistics per benchmark. Reward statistics computed over fusion nodes only.

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	Patience	Patience	Patience	Patience	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%	84.9%
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%	84.9%
Δ (Fusion - Init)	$\pm$0.0%	+5.3%	+1.0%	+2.3%	+2.1%	$-$0.9%
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%

Early Stopping Effectiveness. Four of six benchmarks triggered early stopping (patience=5, min_delta=0.0001), demonstrating that the optimization process efficiently identifies convergence. The earliest stops occur on HumanEval and MBPP (iteration 25), followed by MATH (27) and GSM8K (28). In contrast, HotpotQA and DROP required the full 30 iterations (max-iteration budget), suggesting these tasks benefit from extended exploration due to their more complex reasoning and format constraints.

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 12 presents detailed percentile analysis of workflow scores across all nodes.

Table 12. Score distribution percentiles across all nodes (initial + fusion).

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%	84.9%

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

Table 13. 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}	{79.3,78.5,34.6,84.9,75.6}%	18	7

Direct-Parent Workflow and Prompt Contributions. To make the lineage in Table 13 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.