MoFlow: Multi-Objective Agentic Workflow Generation

Anonymous authors
Paper under double-blind review

Abstract

We study the generation of agentic workflows that jointly optimize multiple objectives, such as accuracy, cost, latency, robustness, and consistency. Existing methods for workflow generation typically optimize accuracy alone or a weighted sum of objectives, so each trained generator commits to one fixed trade-off and must be retrained from scratch when preferences change. To alleviate this, we propose MoFlow, which generates workflows optimized across varied preferences. Specifically, MoFlow formulates workflow generation as a multi-objective Markov decision process and solves it by leveraging Convex-Hull Monte Carlo Tree Search with optimistic set-valued backups, where every node stores a set of reachable trade-offs rather than one weighted score. A single search thus approximately covers the Pareto front, from which MoFlow can return a workflow for any preference by lookup without retraining. We evaluate MoFlow against six strong baselines on six benchmarks spanning mathematics, code, and question answering. Since the baselines are single-scalar optimizers by design, an apples-to-apples comparison is difficult. We instead adopt an evaluation setup that favors the baselines, in that they are rerun for each testing preference, which MoFlow never sees. Even under this stringent setup, MoFlow achieves the highest average hypervolume.[1]

1 Introduction

An agentic workflow, a structured graph that composes LLM calls and task-specific operations, can substantially outperform a single prompt on complex, reasoning-heavy tasks (Yue et al., 2025; Li et al., 2026; Nielsen et al., 2026). A growing body of work therefore automates workflow construction by searching over candidate graphs and optimizing them for a single performance metric such as accuracy, as in AFlow (Zhang et al., 2025d) and its successors (Wang et al., 2025b; Zhao et al., 2025; Chen et al., 2026; Zhang et al., 2026a; Hu et al., 2026). Real-world deployment, however, is inherently multi-objective: beyond accuracy alone, workflows often need to be cheap, fast, robust to noisy input, and consistent across repeated runs (Rabanser et al., 2026).

These objectives often compete, and different real-world deployments may prioritize them differently. For instance, an enterprise workflow prefers low cost and latency, a scientific one robustness and consistency. The right weighting is rarely known in advance, so a practical generator needs to discover a diverse set of trade-offs from which users can choose according to their preferences. Yet no existing generator directly supports this goal. Accuracy-driven generators return a workflow optimized for a single objective (Zhang et al., 2025d; Hu et al., 2025; Wei et al., 2026; Zhang et al., 2026a;b), while the few that look beyond accuracy collapse a handful of objectives, typically accuracy and cost, into a single scalar by simple weighted scalarization (Kong et al., 2026; Zhang et al., 2025b; Xu et al., 2025; Hu et al., 2026). Either way the generator commits to one point on the Pareto front, and reaching a different point requires retraining the generator from scratch. We therefore ask: *can a workflow generator span the Pareto front of workflows in a single run?*

Our answer is yes: rather than settle on one trade-off, a search can remember every trade-off worth making. Representative generators such as AFlow (Zhang et al., 2025d) run Monte Carlo Tree Search (MCTS) (Kocsis & Szepesvári, 2006; Browne et al., 2012; Silver et al., 2016; Schrittwieser et al., 2020) with a scalar per node, collapsing all trade-offs into one number, and generate

[1] Hypervolume measures how well a set of solutions covers the achievable trade-offs among objectives. We open source our code at anonymous.4open.science/r/MoFlow-anonymous-7CE3.

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

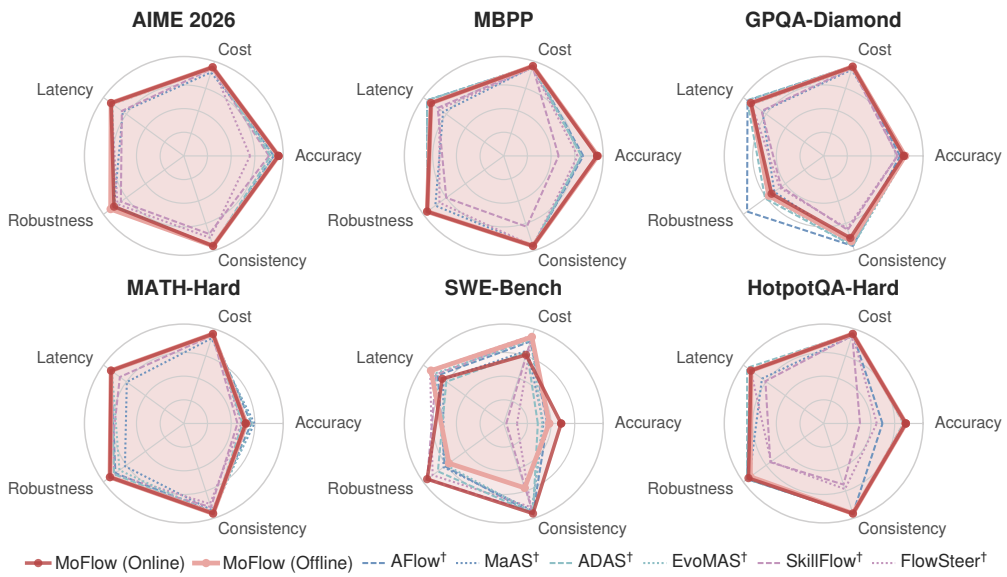


Figure 1: **Per-objective coverage on the six benchmarks: across eleven testing preferences, MoFlow matches or beats every baseline on all five objectives on most tasks.** Each axis shows the best value a method attains across its eleven generated workflows, with cost and latency inverted so that outward is always better. Each baseline† is rerun once per testing preference, whereas MoFlow serves all eleven preferences from one search that never saw them.

each workflow as a whole, making individual design choices hard to trace. We therefore propose **MoFlow**, which keeps the MCTS search but replaces each node's scalar with a convex coverage set (CCS) (Painter et al., 2020), the best trade-offs attainable under each preference, and grows each workflow through stepwise graph edits rather than whole-workflow revision. Formally, we formulate workflow generation as a multi-objective Markov decision process (MOMDP) (Chatterjee et al., 2006), under which Convex-Hull Value Iteration (CHVI) (Barrett & Narayanan, 2008) propagates the per-node CCSs to the root, so a single search approximates a Pareto front across preferences.

Replacing scalars with sets dictates what the method searches for and stores, but not how it allocates its effort. One challenge is where in preference space to search. Each MoFlow trial samples a user preference as a context and, through Contextual Zooming for Trees (CZT) (Slivkins, 2014), conditions action selection on it at every node, so a single tree learns how the best action shifts with the preference. The other challenge is that a partial workflow cannot be run, yet the search must evaluate it to decide where to explore next. Motivated by evidence that workflow performance can be predicted without execution (Zhang et al., 2025e), MoFlow trains a graph neural network (GNN) on executed workflows to evaluate partial ones directly. How the GNN is fit yields the two variants we report throughout. Online MoFlow fits the GNN during the search on workflows it executes, requiring no prior data (more general but more expensive). Offline MoFlow pretrains the GNN on executions from previous online runs and searches with no execution at all (cheaper but less general).

Compared with six strong baselines across six benchmarks spanning mathematics, code, and question answering, on five objectives (we want a workflow to be accurate, consistent, robust, yet cheap and fast), the two MoFlow variants attain the best hypervolume on four benchmarks and trail the best by under 1% on the other two. Averaged over the six benchmarks, both MoFlow variants rank first and second, ahead of every baseline, improving the best baseline's hypervolume by 10.9% (online) and 12.8% (offline, at least 17 times cheaper). The comparison deliberately favors the baselines. None was built for more than two objectives, so we adapt each to optimize the scalarized reward over all five, rerunning it once per testing preference so that every baseline learns the exact weights it is tested on.[2] MoFlow, by contrast, sees none of the testing preferences. Even so, it matches or beats every baseline on each of the five objectives across most benchmarks, as Figure 1 previews.

In summary, we (i) formulate workflow generation as a stepwise MOMDP where an LLM proposes and realizes each graph edit (§3.1); (ii) solve it with a multi-objective tree search whose nodes

[2]Hereafter, † denotes an adapted baseline, which sees the testing preference weights during training.

store CCSs, so a single search approximates a Pareto front that serves any preference (§3.2); (iii) make the search practical by conditioning action selection on the sampled preference via CZT and valuing partial workflows with an uncertainty-gated GNN (§3.3); and (iv) validate MoFlow against six strong baselines across six benchmarks, five objectives, and four base models (§5).

2 Related Work

Table 1: **Comparison of MoFlow with the baselines.** $2\to1$: two objectives scalarized into one. D: number of objectives ($D{=}5$ in our experiments). *Pareto*: one run returns the Pareto front. *Stepwise*: the search builds workflows step by step. E is the cost of one full workflow evaluation (many LLM calls) and ℓ that of a single call ($\ell \ll E$), over N search iterations, N_{tr} training episodes, or horizon H. ‡Plus LLM-policy gradient updates. *Inference*: cost to serve a new preference (a weight vector over the objectives) by rerunning the search or training. MoFlow instead answers any preference by lookup. EvoMAS has no training phase. Its search runs at test time as queries arrive, so a new preference reruns it in full. ✓ yes, ✗ no, ◗ partial.

Method	Family	Objectives	Pareto	Stepwise	Training	Inference
AFlow (Zhang et al., 2025d)	Monolithic	1	✗	✗	$O(NE)$	$O(NE)$ re-search
MaAS (Zhang et al., 2025b)		$2\to1$	✗	✗	$O(N_{\text{tr}}E)$	$O(N_{\text{tr}}E)$ retrain
FlowSteer (Zhang et al., 2026a)	Constructive	1	✗	✓	$O(N_{\text{tr}}H\ell)^{\ddagger}$	$O(N_{\text{tr}}H\ell)$ retrain
SkillFlow (Zhang et al., 2026b)		1	✗	◗	$O(N_{\text{tr}}H\ell)^{\ddagger}$	$O(N_{\text{tr}}H\ell)$ retrain
EvoMAS (Hu et al., 2026)	Evolutionary	$2\to1$	✗	✗	–	$O(NE)$ re-search
ADAS (Hu et al., 2025)		1	✗	✗	$O(NE)$	$O(NE)$ re-search
MoFlow (Online)	Constructive	D	✓	✓	$O(NE)$	lookup (no LLM)
MoFlow (Offline)		D	✓	✓	$O(NH\ell)$	

2.1 Automatic Workflow Generation

We organize automatic workflow generators into three families (Table 1). MoFlow does not sit squarely in any of these three categories: it is polylithic, not monolithic (though it is MCTS-based as in AFlow), constructive but multi-objective (unlike other constructive methods), and learning-based rather than evolutionary.

Monolithic workflow generation. A first line of work treats the complete workflow as the unit of generation and revision, scoring each candidate by executing it end to end. The most representative is AFlow (Zhang et al., 2025d), which also applies MCTS: each tree node holds a full workflow, and each expansion invokes an LLM optimizer to revise that workflow as a whole. Executing the revised workflow then gives the node a single accuracy score. The search thus returns one workflow rather than a Pareto front, and credit goes to the entire revision, never to the individual edits within it. Both limitations extend across the family, whether a method trains against a scalar reward (Fan et al., 2024; Wang et al., 2025b; Gao et al., 2025; Xu et al., 2026), samples from a learned architecture distribution as MaAS does (Zhang et al., 2025b), or composes reusable operators (Zhao et al., 2025).

Constructive workflow generation. A more recent line addresses credit assignment by constructing workflows through stepwise decisions. FlowSteer (Zhang et al., 2026a) builds a workflow through atomic canvas edits, SkillFlow (Zhang et al., 2026b) through stepwise invocations of recursively evolving skills, and related methods likewise assign credit or capabilities to individual steps (Zhuge et al., 2024; Niu et al., 2025; Chen et al., 2026; Kong et al., 2026; Wang et al., 2026a; Yuan et al., 2026). Although this finer granularity improves interpretability, it does not by itself preserve trade-offs among objectives. A few methods do consider a second objective beyond accuracy, either token cost (Wang et al., 2025c; Zhang et al., 2025c) or robustness (Xu et al., 2025), yet all of them collapse the two into a single scalar. In contrast, we keep the stepwise search but propagate each node's CCS with CHVI backups, so each edit receives separate credit along every objective.

Evolutionary workflow generation. A third line grows workflows by iterative variation and selection rather than by planning (Zhang et al., 2025a; Wang et al., 2025a; Liu et al., 2026; Xiao et al.,

2026). EvoMAS (Hu et al., 2026) mutates and recombines configurations, while ADAS (Hu et al., 2025) has a meta agent write new designs in code, building on a growing archive of prior ones. These methods explore a rich space of complete workflows, but a single scalar still decides what survives each round, so the search again converges to one region of the front.

Two additional lines of work underpin MoFlow directly. Rabanser et al. (2026) propose four dimensions for the scientific evaluation of AI agents. We adopt two of them (robustness and consistency) alongside accuracy, cost, and latency to form our five objectives. Zhang et al. (2025e) argue that workflow performance is predictable from the workflow graph and its prompts alone, without execution, an idea extended by Guan et al. (2025) and Feng et al. (2026). Because partial workflows cannot yet be executed, we use a GNN to evaluate them directly from their graph representation.

2.2 Tree Search for Multi-Objective Reinforcement Learning

Solving an MOMDP means recovering a Pareto front of policies (Roijers et al., 2013; Yang et al., 2019; Alegre et al., 2023; Basaklar et al., 2023; Zhang et al., 2023; Liu et al., 2025a; Li et al., 2025; Liu et al., 2025b), and multi-objective MCTS (MOMCTS) pursues this goal by adaptively allocating simulations across the front. The original MOMCTS (Wang & Sebag, 2012) keeps a single global front at the root, which cannot express purely local trade-offs (Painter et al., 2020; Hayes et al., 2022). Later methods instead maintain a local Pareto set at every node, following Pareto Q-learning (Moffaert & Nowé, 2014), and differ in action selection: Pareto dominance (Chen & Liu, 2019; Drugan & Nowe, 2013), hypervolume-based UCB (Auer et al., 2002; Perez et al., 2013; 2015; Perez-Liebana et al., 2016), or Chebyshev scalarization (Xu & Li, 2017). All of these assume deterministic transitions, where the chosen action determines the successor, whereas ours are stochastic because an LLM can instantiate the same action into different workflows. We build on Convex-Hull MCTS (CHMCTS) (Painter et al., 2020), which already handles stochastic transitions via CHVI within Trial-based Heuristic Tree Search (Keller & Helmert, 2013), and extend it with an LLM that proposes and realizes actions, so neither the action set nor the transition model is fixed.

3 MoFlow: Multi-Objective Agentic Workflow Generation

3.1 Workflow Construction as a Multi-Objective MDP

Building a workflow requires a sequence of actions whose outcomes may trade off against one another, so we formalize it as a finite-horizon multi-objective Markov decision process (MOMDP) $M = \langle S, A, R, T, \bar{s}, H \rangle$. An MOMDP replaces the scalar reward of an ordinary MDP with a vector of D objectives, $R : S \times A \times S \to \mathbb{R}^D$. In our setting, this reward is obtained only at the terminal state by executing the complete workflow, and we cast every objective as maximization, so cost and latency are negated in R. The remaining elements are standard: S is the state space of partial workflows, A the action space of atomic edits, T the transition kernel, $\bar{s}$ the empty workflow, and H the horizon. Because rewards are vectors, a policy π is evaluated by a vector value $\boldsymbol{v}^\pi(s) = \mathbb{E}\big[\sum_t R(s_t, a_t, s_{t+1}) \mid s_0 = s, \pi\big] \in \mathbb{R}^D$, and a user *preference* is a weight vector $\boldsymbol{w} \in \Delta^{D-1}$ that scalarizes it as $\boldsymbol{w}^\top \boldsymbol{v}$. Two value vectors need not be comparable (one workflow may be more accurate while another is cheaper), so solving an MOMDP means ranking outcomes by *Pareto dominance* and recovering the entire *Pareto front* of non-dominated trade-offs in one run.

Definition 1 (Pareto dominance and Pareto front)**.** *A vector $\boldsymbol{u}$* dominates *$\boldsymbol{v}$, $\boldsymbol{u} \succ \boldsymbol{v}$, if $u_i \geq v_i$ for every objective i and $u_j > v_j$ for at least one j. Given a set $Y \subset \mathbb{R}^D$ of achievable value vectors, its* Pareto front *is the non-dominated subset $P = \{\, \boldsymbol{r} \in Y : \nexists\, \boldsymbol{r}' \in Y \text{ with } \boldsymbol{r}' \succ \boldsymbol{r} \,\}$.*

We measure the quality of a front P by its *hypervolume*, the volume of objective space that P dominates above a fixed reference point $\boldsymbol{r}_0$,

$$\mathrm{HV}(P; \boldsymbol{r}_0) = \Lambda\big(\{\, \boldsymbol{y} \in \mathbb{R}^D : \exists \boldsymbol{p} \in P,\ \boldsymbol{r}_0 \preceq \boldsymbol{y} \preceq \boldsymbol{p} \,\}\big),$$

with Λ the Lebesgue measure and larger better. Since we only consider linear preferences, which scalarize the value vector as $\boldsymbol{w}^\top \boldsymbol{v}$, it suffices to keep a *convex coverage set* (CCS), a subset $\mathrm{CCS} \subseteq P$ that still attains the best scalarized value under every preference, $\max_{\boldsymbol{v} \in \mathrm{CCS}} \boldsymbol{w}^\top \boldsymbol{v} = \max_{\boldsymbol{v} \in Y} \boldsymbol{w}^\top \boldsymbol{v}$ for all $\boldsymbol{w} \in \Delta^{D-1}$ (Roijers et al., 2013). Our backups prune only Pareto-dominated vectors, so each node in fact stores the full front, a superset of the minimal CCS (Appendix C.2).

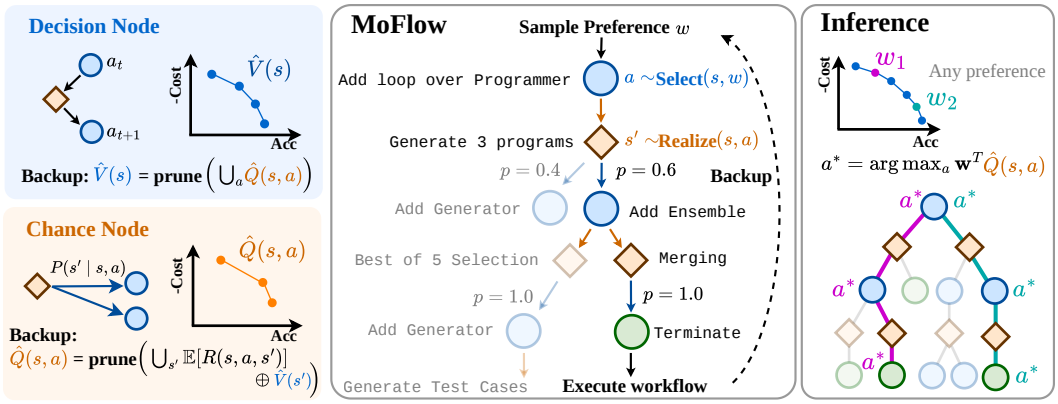


Figure 2: **Overall MoFlow framework.** MoFlow searches once, then answers any preference by lookup. Each trial samples a preference $\boldsymbol{w}$ and walks the tree as follows. At a **decision node**, the trial selects an action for $\boldsymbol{w}$; at a **chance node**, an LLM realizes that action into at most N_r distinct outcomes, each visited with empirical probability p estimating $T(s' \mid s, a)$. Decision and chance nodes alternate until a **terminal** action completes the workflow; MoFlow then executes it and backs the result up the visited path (faded nodes are branches the current trial does not visit). Every node therefore stores a convex coverage set ($\hat{V}(s)$ at decision nodes, $\hat{Q}(s,a)$ at chance nodes), so a single tree can unroll different workflows for different preferences such as $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ (*Inference* panel).

Algorithm 1 Algorithm of MoFlow for a single trial.

$\boldsymbol{w} \sim \Delta^{D-1}$ ▷ the CZT context, resampled at each trial
$s \leftarrow \bar{s}$
while s is not terminal **and** s is already in the tree **do** ▷ a trial ends at a terminal or newly reached state
  $\widehat{A} \leftarrow$ PROPOSE(s); $\mathcal{B} \leftarrow$ the relevant balls of $\widehat{A}$ at $\boldsymbol{w}$ ▷ at the decision node s
  $a \leftarrow$ SELECT$(s, \boldsymbol{w})$, the action of $\arg\max_{B \in \mathcal{B}} I(B)$ ▷ Eq. 3
  $s' \sim$ REALIZE(s, a); update frequency p; $s \leftarrow s'$ ▷ at the chance node (s, a)
end while
$\hat{V}_{\text{leaf}} \leftarrow$ execute s if terminal, else $f_\theta(s)$; $u \leftarrow \max_{\boldsymbol{v} \in \hat{V}_{\text{leaf}}} \boldsymbol{w}^\top \boldsymbol{v}$
for (s, a, B, s') along the visited path, back to the root **do**
  $\hat{Q}(s,a) \leftarrow$ Eq. 1; $\hat{V}(s) \leftarrow$ Eq. 2 ▷ CHVI backup
  fold u into $\nu(B)$; split B at $\boldsymbol{w}$ if $\text{conf}(B) \le r(B)$ ▷ the contextual zooming step
end for

3.2 MoFlow Search and Inference

MoFlow solves the MOMDP by MCTS search in which each node is valued by the set of trade-offs reachable from it rather than by a single scalar. Because each trial samples a fresh preference $\boldsymbol{w}$ to guide action selection, a single search approximately covers the front. Given any preference $\boldsymbol{w}$ (even one never sampled during search), the resulting tree serves the preference by lookup without retraining. Algorithm 1 gives an overview of the single trial procedure.

The search is driven by two LLM roles. A *proposer* suggests at most b candidate edits at each partial workflow, and a *realizer* instantiates the selected edit into at most N_r distinct outcomes. The tree accordingly alternates two kinds of node: a *decision node*, where the search chooses among the proposer's suggested actions, and a *chance node*, where it observes which successor the realizer produces. Both store a CCS of value vectors, preserving the trade-offs among objectives rather than collapsing them to a scalar. Figure 2 illustrates these sets with two objectives, accuracy and cost.

Definition 2 (Decision node)**.** *A* decision node *is a partial workflow $s \in S$ that stores a CCS estimate $\hat{V}(s)$. On each visit, the proposer returns at most b candidate actions drawn from an operator pool, and the search selects one for the current preference (§3.3). A child chance node is created for the selected action. The root is the empty workflow $\bar{s}$.*

Definition 3 (Chance node)**.** *A* chance node *is a state-action pair $(s, a) \in S \times A$ that stores a CCS estimate $\hat{Q}(s, a)$. Because the realizer may produce a different successor on each visit, a single chance node can branch into multiple child decision nodes s', and each branch's empirical visit frequency p estimates $T(s' \mid s, a)$ for sampling at inference.*

Separating the chosen action from its realization turns stochasticity into counted branches rather than score noise. CHVI maintains both CCS estimates by backing up the entire front after each trial. Its exact chance backup is the probability-weighted Minkowski sum over successors, which we replace with a cheaper *optimistic* union that keeps every vector achievable under at least one realization:

$$\hat{Q}(s,a) = \mathrm{prune}\Big(\bigcup_{s'} \big(\mathbb{E}[R(s,a,s')] \oplus \hat{V}(s')\big)\Big), \tag{1}$$

where prune keeps only the non-dominated vectors. A workflow can only be run once it is complete, so $\mathbb{E}[R]$ is estimated only at terminal states, from repeated executions of the completed workflow. A partial workflow carries no reward and contributes only $\hat{V}(s')$, estimated by a GNN value model introduced below. For a decision node, CHVI unions and prunes its actions' CCS estimates:

$$\hat{V}(s) = \mathrm{prune}\Big(\bigcup_{a} \hat{Q}(s,a)\Big). \tag{2}$$

Once a trial reaches its leaf and the completed workflow is evaluated, MoFlow applies the two backups in alternation back up the visited path to the root. Each node's CCS then covers the best trade-offs its completions can achieve, so a single trial improves the answer for every preference, not just the one it sampled. Appendix C.2 explains why the chance backup uses an optimistic union rather than the exact probability-weighted sum.

When the leaf is still partial, some objectives require execution to evaluate. A GNN value model f_θ predicts these objectives along with its own uncertainty. If the uncertainty exceeds a gate, MoFlow executes a few cheap completions and replaces the prediction with the non-dominated returns (details in Appendix D). These returns also fill a replay buffer that online MoFlow uses to update f_θ as the search proceeds. Offline MoFlow instead pretrains f_θ on executions logged from online runs, then searches without any execution.

At inference, given a preference $\boldsymbol{w}$, MoFlow walks the built tree from the root, selecting the $\boldsymbol{w}$-optimal action $a^* = \arg\max_a \max_{\boldsymbol{v}\in\hat{Q}(s,a)} \boldsymbol{w}^\top \boldsymbol{v}$ at each decision node and sampling a successor by its empirical visit frequency p at each chance node (*Inference* panel of Figure 2). The entire walk reads stored CCS estimates, so serving a new preference requires no additional LLM calls.

3.3 Contextual Zooming for Trees

Preference enters the search only at action selection. Each decision node is a contextual bandit (Abbasi-yadkori et al., 2011) whose arms are the proposed actions and whose winning arm changes across the simplex. A single ranking of the arms for all $\boldsymbol{w}$ incurs regret $\Omega(N)$ over N trials (Painter et al., 2020), and a uniform grid over Δ^{D-1} achieves resolution everywhere at a cost exponential in D. Contextual Zooming for Trees (CZT) (Painter et al., 2020) instead stays coarse where one action dominates and refines only near the boundaries where the winner changes. Its regret scales with the zooming dimension c of those boundaries rather than the ambient dimension $D-1$.

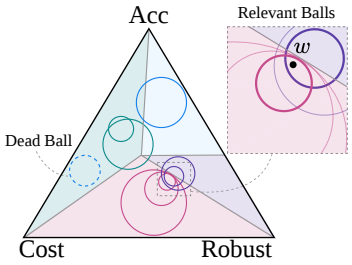


Figure 3: **Contextual zooming at one decision node** over the preference simplex for D=3 objectives. Each ball ties one action (its color) to a preference region. The background marks the true optimal action the search never sees. Finer balls tell nearby preferences apart, so refinement drifts toward region boundaries while interiors stay coarse.

Concretely, CZT covers the preference simplex with balls, each tying one action to a preference region (Figure 3). Only the finest ball of each legal action covering the sampled $\boldsymbol{w}$ is *relevant*, and these compete by the optimistic index:

$$I_{\mathrm{pre}}(B) = \underbrace{\nu(B)}_{\text{reward}} + \underbrace{r(B)}_{\text{radius}} + \underbrace{\mathrm{conf}(B)}_{\text{confidence}},$$
$$I(B) = r(B) + \min_{B' \text{ active}} \big[I_{\mathrm{pre}}(B') + d_s(B,B')\big]. \tag{3}$$

An action is more likely to be selected when its relevant ball B has a high reward $\nu(B)$ (the average scalarized reward under the sampled $\boldsymbol{w}$), few visits $n(B)$ (a wide $\mathrm{conf}(B) = \kappa\sqrt{\log N/(1+n(B))}$, with N the total trial budget), or a coarse radius $r(B)$. Once a ball has been selected enough times to be confident at its own scale (i.e., $\mathrm{conf}(B) \le r(B)$), we split it into a half-radius child centered at

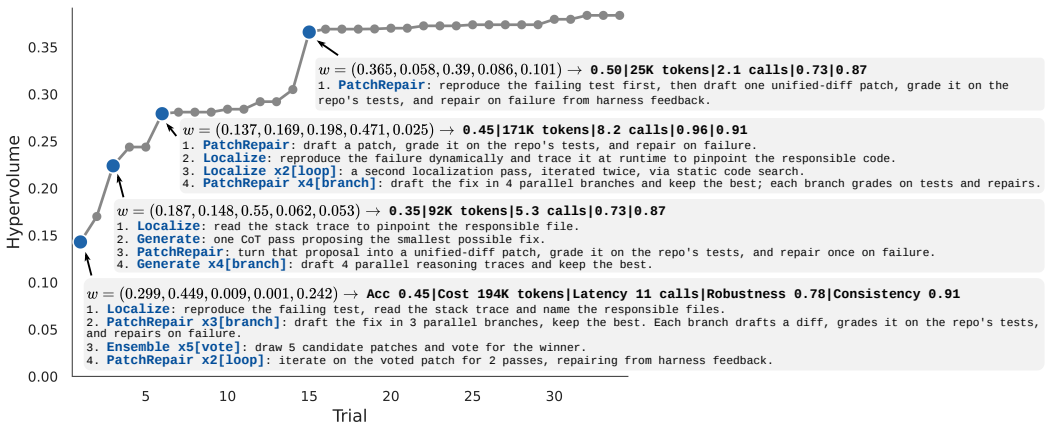


Figure 4: **One MoFlow search on SWE-Bench.** The y-axis shows the in-sample hypervolume of non-dominated workflows executed up to each trial. Each labeled example is a workflow generated from its sampled $\boldsymbol{w}$, with objectives listed as accuracy | cost | latency | robustness | consistency.

the current $\boldsymbol{w}$. A ball whose action keeps losing at the preferences it covers eventually stops being selected and goes *dead*. Unlike classical CHMCTS, our proposer redraws the action set at every visit. Because the operator pool is fixed, the same operators reappear across visits but with different free-text roles, adding diversity without creating blind spots (Appendix C).

4 Experimental Setup

Benchmarks. We evaluate on six benchmarks spanning three task families for which agentic workflows are typically built. For mathematics we use AIME 2026 (Dekoninck et al., 2026) and MATH-Hard, the Level-5 subset of MATH (Hendrycks et al., 2021). For code we use the sanitized subset of MBPP (Austin et al., 2021) and SWE-Bench Lite (Jimenez et al., 2024). For question answering we use GPQA-Diamond (Rein et al., 2024) and HotpotQA-Hard (Yang et al., 2018). Each benchmark is split into a build set used during search and a disjoint held-out set on which we report all results (10 build / 20 held-out for AIME 2026, $20/50$ elsewhere).[3]

Objectives and metrics. Each workflow is evaluated on five objectives: accuracy, cost, latency, robustness, and consistency. We execute every problem six times, with $J = 3$ repeated runs of the original problem and one run of each of its $m = 3$ paraphrases. We report accuracy (task success), cost (total tokens per problem), and latency (sequential LLM calls) by averaging over the repeated runs. Robustness and consistency share the formula $\mathrm{clip}(1 - 4\mathrm{Var}(\mathrm{acc}), 0, 1)$ and differ only in where the accuracy variance comes from. Consistency measures it across the repeated runs, while robustness measures it across the paraphrases. A good workflow attains high accuracy, robustness, and consistency at low cost and latency.[4] Each method generates one workflow per testing preference, and we evaluate it by the hypervolume of these workflows on the held-out set with respect to the nadir point $\boldsymbol{r}_0 = [0, -3\times10^5, -20, 0, 0]$.

Baselines. We compare against six recent workflow generators, two from each family in §2: AFlow (Zhang et al., 2025d), MaAS (Zhang et al., 2025b), FlowSteer (Zhang et al., 2026a), SkillFlow (Zhang et al., 2026b), ADAS (Hu et al., 2025), and EvoMAS (Hu et al., 2026). All methods are evaluated on the same eleven testing preferences $\boldsymbol{w}$, spanning the simplex from corners to center: the five permutations of $[1, 0, 0, 0, 0]$, the five permutations of $[0.6, 0.1, 0.1, 0.1, 0.1]$, and the uniform vector $[0.2, \ldots, 0.2]$. All six baselines are designed for single-scalar optimization, so we adapt each to optimize $\boldsymbol{w}^\top \boldsymbol{v}$ over all five objectives, with a new run per preference. **This protocol**

[3]Each workflow runs six times per problem, so evaluating MoFlow on a 50-problem held-out set takes up to 3,300 executions (over 10,000 LLM calls on SWE-Bench alone), already matching AFlow's total evaluation budget. The 10–20 problem build sets still suffice for generalization, as shown in Table 5 (Appendix A.2).

[4]The five objectives capture distinct properties. Even the closest pair, robustness and consistency, correlates at only 0.63 over 132 served workflows, from -0.13 on GPQA to 0.87 on SWE-Bench. We measure latency as sequential LLM calls rather than wall-clock time because it varies with provider, hardware, and load.

Table 2: **Held-out hypervolume of the front served across eleven testing preferences.** All methods use GPT-5 Mini. Each baseline† is adapted to the five objectives and rerun once per preference, targeting the exact weight it is later scored at. MoFlow instead serves all eleven preferences from a single search that never saw them. **Even so, MoFlow attains the best hypervolume on four of six benchmarks and the best mean. Bold** marks the best per column, underline the second best. Mean is reported with standard deviation across the six benchmarks.

Method	AIME 2026	MATH-Hard	MBPP	SWE-Bench	GPQA-D	HotpotQA-H	Mean
AFlow†	29.85	**21.87**	27.71	7.11	17.41	18.57	20.42 ±8.17
MaAS†	24.61	14.28	21.37	7.00	15.13	24.69	17.85 ±6.97
FlowSteer†	18.47	17.34	19.86	4.06	12.58	9.68	13.66 ±6.07
SkillFlow†	18.55	15.75	10.77	0.81	10.35	4.89	10.19 ±6.59
ADAS†	27.71	19.84	26.50	6.84	20.17	28.89	21.66 ±8.22
EvoMAS†	28.68	20.74	22.59	7.30	20.14	**30.35**	21.63 ±8.19
MoFlow (Online)	31.39	21.69	34.92	**7.54**	18.42	30.14	24.02 ±10.19
MoFlow (Offline)	**33.02**	21.85	**35.14**	6.89	**20.39**	29.27	**24.43** ±10.42

deliberately favors the baselines. Each baseline targets the exact weights it is scored at, whereas MoFlow never sees them during search and serves all eleven from one tree.

Models and prompts. We test MoFlow and all six baselines on GPT-5 Mini (Singh et al., 2026), Claude Haiku 4.5 (Anthropic, 2025), Gemini 3 Flash Preview (DeepMind, 2025), and DeepSeek-V4-Flash (DeepSeek-AI et al., 2026). Within a run, a single model makes every LLM call: it proposes at most $b = 4$ actions on each visit to a decision node, realizes the selected action into at most $N_r = 3$ role specializations, and executes each finished workflow at temperature 0.7.

5 Experimental Results and Analysis

Example. Figure 4 shows one MoFlow search on SWE-Bench building a front of workflows. Each trial samples a fresh preference $\boldsymbol{w}$ and rolls out one workflow, so the hypervolume of the non-dominated set rises whenever a new trade-off enters the front. The front ranges from a single `PatchRepair` at 25k tokens to a four-step workflow that branches, votes, and iteratively repairs at 194k. Every construction trace records how each action changed the five objectives, so practitioners can see which actions played the most significant role in shaping the final performance.

Main results. Table 2 reports the hypervolume of the front each method serves across the eleven testing preferences. MoFlow is strongest overall, with its two variants achieving the two best mean hypervolumes.[5] We attribute this advantage to cross-preference transfer. A baseline improves its solution only for the preference it optimizes. MoFlow also searches under one sampled preference per trial, but CHVI backs up each outcome as a vector available to all preferences, so every preference benefits from the entire search.

Online and offline MoFlow. The two MoFlow variants in Table 2 reach nearly the same hypervolume at very different search costs. Online MoFlow executes every generated workflow, so execution accounts for over 99% of search tokens: a single search spends 132.3M execution tokens on AIME 2026 and 134.9M on MBPP (Table 5), while the baselines average 5.6–28.7M per preference. Offline MoFlow instead pretrains f_θ on the execution data those online searches already collected across the six tasks, so its search runs no workflow and costs only 2.9–3.6M proposer and realizer tokens, about 2% of the online cost and a significant saving compared with the baselines. Yet the offline version attains slightly better hypervolume, showing that once enough execution data is available, MoFlow can search effectively with a pretrained evaluator.

MoFlow outperforms on nearly every objective, not just in hypervolume. Hypervolume summarizes all five objectives into one score, so Figure 1 breaks the comparison out by objective. Each

[5]Hypervolume magnitude depends on the reference point $\boldsymbol{r}_0$ but the *relative* ranking across methods provides an unbiased comparison.

Table 3: **Held-out hypervolume under four base models, where MoFlow is the only method that ranks in the top two on all four.** Each method is rerun end to end with the tested model.

Method	GPT-5 Mini	Haiku 4.5	Gemini 3 Flash	DeepSeek-V4-Flash	Mean	Worst
AFlow†	<u>29.85</u>	14.54	**30.33**	19.86	23.64 ±7.75	14.54
MaAS†	24.61	14.40	17.88	17.77	18.66 ±4.28	14.40
FlowSteer†	18.47	15.78	28.75	13.61	19.15 ±6.70	13.61
SkillFlow†	18.55	**16.80**	2.39	14.68	13.11 ±7.32	2.39
ADAS†	27.71	16.02	29.19	**22.90**	<u>23.95</u> ±5.93	<u>16.02</u>
EvoMAS†	28.68	7.60	10.83	15.33	15.61 ±9.27	7.60
MoFlow (Online)	**31.39**	<u>16.09</u>	<u>30.15</u>	<u>22.13</u>	**24.94** ±7.19	**16.09**

axis shows the best value a method attains across the eleven testing preferences. MoFlow matches or beats every preference-tuned baseline on nearly every objective and benchmark.

5.1 Ablation Study

MoFlow is generalizable across models. Table 3 reruns the AIME 2026 comparison under three additional base models. MoFlow ranks first on GPT-5 Mini and second on the other three and the best overall (the mean). No other method stays in the top two on all four. Each baseline that leads one column falls behind on another: SkillFlow leads on Claude Haiku 4.5 but collapses to 2.39 on Gemini 3 Flash Preview, AFlow leads on Gemini but falls 2.27 below MoFlow on DeepSeek-V4-Flash, and ADAS leads on DeepSeek but trails elsewhere. In practice, deployments fix the base model for reasons unrelated to workflow quality, so the worst-case score matters most, and MoFlow ranks first there (16.09).

Preference-conditioned selection matters. We replace CZT at the decision node with the three other action-selection rules of §2.2, leaving every other component of MoFlow unchanged. Hypervolume-UCB and Pareto-UCB each reduce an action's value set to a single scalar that ignores the sampled preference. Chebyshev scalarization does condition on $\boldsymbol{w}$, but keeps one global estimate per action across the entire simplex. All three lose 12–17% of the in-sample hypervolume (on the build set under $\boldsymbol{r}_0$) while spending more tokens. CZT keeps separate estimates per simplex region and refines only where the winning action changes, yielding a better front at lower cost.

Table 4: **Action-selection ablation on AIME 2026 with Claude Haiku 4.5.** CZT attains the best in-sample hypervolume at the lowest cost.

Action selection	In-sample HV ↑	Tokens (M) ↓
Chebyshev	17.06	43.76
Hypervolume-UCB	17.87	61.01
Pareto-UCB	16.80	26.44
CZT (ours)	**20.35**	**23.99**

Query-level search does not generate a better workflow than task-level search. MoFlow also supports query-level search, which builds a small tree per query instead of one tree per task. We run both modes on all six benchmarks under the same setting. Building the twenty query-level trees costs 0.6–0.9× the execution tokens of one task-level tree, yet the workflows they produce consume 1.1–11.6× the tokens and 1.1–6.6× the calls per problem, with no gain on other objectives. The reason is that query-level search is more likely to overfit, with a held-out to in-sample hypervolume ratio as low as 0.28, compared with at least 0.66 for task-level trees (Table 5). Query-level search is thus unnecessary. Our finding echoes the recent finding of Wang et al. (2026b), who show that a small set of top-ranked task-level workflows already covers as many queries as a per-query generator.

6 Conclusion

We presented MoFlow, a multi-objective workflow generator that formulates workflow generation as an MOMDP and solves it with Convex-Hull MCTS. Its core idea is to value every node by a set of reachable trade-offs rather than a scalar, so a single search approximates the Pareto front and serves any preference by lookup. Across six benchmarks and four base models, MoFlow consistently ranks in the top two in average hypervolume. We envision MoFlow as a foundation for agentic systems that replace one-size-fits-all pipelines with a Pareto landscape of trade-offs, searched once and queried at inference time for each user's choice of quality, cost, and trust. Appendix B discusses limitations.

AI USE STATEMENT

We did not use generative AI tools for research ideation, theoretical development, methodology design, or interpretation of results. We used generative AI tools only to polish the wording of the paper for grammar and readability. We have reviewed all AI-polished text and verified that it preserves our intended meaning and claims. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

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

Lucas N. Alegre, Ana L. C. Bazzan, Diederik M. Roijers, Ann Nowé, and Bruno C. da Silva. Sample-efficient multi-objective learning via generalized policy improvement prioritization. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, pp. 2003–2012, Richland, SC, 2023. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450394321. URL https://dl.acm.org/doi/10.5555/3545946.3598872.

Anthropic. Claude haiku 4.5 system card. Technical report, Anthropic, October 2025. URL https://www.anthropic.com/claude-haiku-4-5-system-card.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002. URL https://link.springer.com/article/10.1023/A:1013689704352.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 41–47, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390162. URL https://doi.org/10.1145/1390156.1390162.

Toygun Basaklar, Suat Gumussoy, and Umit Ogras. PD-MORL: Preference-driven multi-objective reinforcement learning algorithm. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=zS9sRyaPFlJ.

BerriAI. LiteLLM. https://github.com/BerriAI/litellm, 2023.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. URL https://ieeexplore.ieee.org/document/6145622.

Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Markov decision processes with multiple objectives. In Bruno Durand and Wolfgang Thomas (eds.), *STACS 2006*, pp. 325–336, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-32288-7. URL https://link.springer.com/chapter/10.1007/11672142_26.

Weizhe Chen and Lantao Liu. Pareto monte carlo tree search for multi-objective informative planning. In *Robotics: Science and Systems XV*. Robotics: Science and Systems Foundation, 2019. doi: 10.15607/rss.2019.xv.072. URL http://dx.doi.org/10.15607/RSS.2019.XV.072.

Xudong Chen, Yixin Liu, Hua Wei, and Kaize Ding. Lemon: Learning executable multi-agent orchestration via counterfactual reinforcement learning, 2026. URL https://arxiv.org/abs/2605.14483.

Google DeepMind. Gemini 3 flash model card. Technical report, Google DeepMind, 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms, 2026. URL https://arxiv.org/abs/2605.00674.

Madalina M. Drugan and Ann Nowe. Designing multi-objective multi-armed bandits algorithms: A study. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–

8, 2013. doi: 10.1109/IJCNN.2013.6707036. URL https://ieeexplore.ieee.org/document/6707036.

Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Workflowllm: Enhancing workflow orchestration capability of large language models, 2024. URL https://arxiv.org/abs/2411.05451.

Xiao Feng, Bo Han, Zhanke Zhou, Jiaqi Fan, Jiangchao Yao, Ka Ho Li, Dahai Yu, and Michael Ng. Rewardflow: Topology-aware reward propagation on state graphs for agentic RL with large language models. In *The 1st Workshop on Scaling Post-training for LLMs*, 2026. URL https://openreview.net/forum?id=qJ284O3B0L.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents, 2025. URL https://arxiv.org/abs/2504.15257.

Wei Guan, Jian Cao, Jinyu Cai, Qiqi Cai, Jianqi Gao, and See-Kiong Ng. Glow: Graph-language co-reasoning for agentic workflow performance prediction, 2025. URL https://arxiv.org/abs/2512.15751.

Conor F. Hayes, Mathieu Reymond, Diederik M. Roijers, Enda Howley, and Patrick Mannion. Monte carlo tree search algorithms for risk-aware and multi-objective reinforcement learning, 2022. URL https://arxiv.org/abs/2211.13032.

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

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

Yuntong Hu, Yuting Zhang, Matthew Trager, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. Evomas: Evolutionary generation of multi-agent systems, 2026. URL https://arxiv.org/abs/2602.06511.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. In *Proceedings of the international conference on automated planning and scheduling*, volume 23, pp. 135–143, 2013. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13557.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pp. 282–293. Springer Berlin Heidelberg, 2006. URL https://link.springer.com/chapter/10.1007/11871842_29.

Mingze Kong, Zikun Qu, Zhongquan Zhou, Pengyu Liang, Xiang Li, Zhiwei Shang, Zhi Hong, Kaiyu Huang, Zhiyong Wang, and Zhongxiang Dai. Workflow-r1: Group sub-sequence policy optimization for multi-turn workflow construction, 2026. URL https://arxiv.org/abs/2602.01202.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23142–23150, 2026. URL https://arxiv.org/abs/2507.18224.

Yining Li, Peizhong Ju, and Ness Shroff. How to find the exact pareto front for multi-objective MDPs? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=S4dItvpvAv.

Erlong Liu, Yu-Chang Wu, Xiaobin Huang, Chengrui Gao, Ren-Jian Wang, Ke Xue, and Chao Qian. Pareto set learning for multi-objective reinforcement learning, 2025a. URL https://arxiv.org/abs/2501.06773.

Ruohong Liu, Yuxin Pan, Linjie Xu, Lei Song, Pengcheng You, Yize Chen, and Jiang Bian. Efficient discovery of pareto front for multi-objective reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=fDGPIuCdGi.

Siwei Liu, Jinyuan Fang, Han Zhou, Yingxu Wang, and Zaiqiao Meng. Sew: Self-evolving agentic workflows for automated code generation, 2026. URL https://arxiv.org/abs/2505.18646.

Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 15(107):3663–3692, 2014. URL http://jmlr.org/papers/v15/vanmoffaert14a.html.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=U23A2BUKYt.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sLKDbuyq99.

Michael Painter, Bruno Lacerda, and Nick Hawes. Convex hull monte-carlo tree-search. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 217–225. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/ICAPS/article/view/6664.

Diego Perez, Spyridon Samothrakis, and Simon Lucas. Online and offline learning in multi-objective monte carlo tree search. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, 2013. doi: 10.1109/CIG.2013.6633621. URL https://ieeexplore.ieee.org/document/6633621.

Diego Perez, Sanaz Mostaghim, Spyridon Samothrakis, and Simon M. Lucas. Multiobjective monte carlo tree search for real-time games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):347–360, 2015. doi: 10.1109/TCIAIG.2014.2345842. URL https://ieeexplore.ieee.org/document/6872573.

Diego Perez-Liebana, Sanaz Mostaghim, and Simon M. Lucas. Multi-objective tree search approaches for general video game playing. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 624–631, 2016. doi: 10.1109/CEC.2016.7743851. URL https://ieeexplore.ieee.org/document/7743851.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4LTwRqhmhk.

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

D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, October 2013. ISSN 1076-9757. doi: 10.1613/jair.3987. URL http://dx.doi.org/10.1613/jair.3987.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. URL https://www.nature.com/articles/s41586-020-03051-4.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. URL https://www.nature.com/articles/nature16961.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, Alex Makelov, Alex Neitz, Alex Wei, Alexandra Barr, Alexandre Kirchmeyer, Alexey Ivanov, Alexi Christakis, Alistair Gillespie, Allison Tam, Ally Bennett, Alvin Wan, Alyssa Huang, Amy McDonald Sandjideh, Amy Yang, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrei Gheorghe, Andres Garcia Garcia, Andrew Braunstein, Andrew Liu, Andrew Schmidt, Andrey Mereskin, Andrey Mishchenko, Andy Applebaum, Andy Rogerson, Ann Rajan, Annie Wei, Anoop Kotha, Anubha Srivastava, Anushree Agrawal, Arun Vijayvergiya, Ashley Tyra, Ashvin Nair, Avi Nayak, Ben Eggers, Bessie Ji, Beth Hoover, Bill Chen, Blair Chen, Boaz Barak, Borys Minaiev, Botao Hao, Bowen Baker, Brad Lightcap, Brandon McKinzie, Brandon Wang, Brendan Quinn, Brian Fioca, Brian Hsu, Brian Yang, Brian Yu, Brian Zhang, Brittany Brenner, Callie Riggins Zetino, Cameron Raymond, Camillo Lugaresi, Carolina Paz, Cary Hudson, Cedric Whitney, Chak Li, Charles Chen, Charlotte Cole, Chelsea Voss, Chen Ding, Chen Shen, Chengdu Huang, Chris Colby, Chris Hallacy, Chris Koch, Chris Lu, Christina Kaplan, Christina Kim, CJ Minott-Henriques, Cliff Frey, Cody Yu, Coley Czarnecki, Colin Reid, Colin Wei, Cory Decareaux, Cristina Scheau, Cyril Zhang, Cyrus Forbes, Da Tang, Dakota Goldberg, Dan Roberts, Dana Palmie, Daniel Kappler, Daniel Levine, Daniel Wright, Dave Leo, David Lin, David Robinson, Declan Grabb, Derek Chen, Derek Lim, Derek Salama, Dibya Bhattacharjee, Dimitris Tsipras, Dinghua Li, Dingli Yu, DJ Strouse, Drew Williams, Dylan Hunn, Ed Bayes, Edwin Arbus, Ekin Akyurek, Elaine Ya Le, Elana Widmann, Eli Yani, Elizabeth Proehl, Enis Sert, Enoch Cheung, Eri Schwartz, Eric Han, Eric Jiang, Eric Mitchell, Eric Sigler, Eric Wallace, Erik Ritter, Erin Kavanaugh, Evan Mays, Evgenii Nikishin, Fangyuan Li, Felipe Petroski Such, Filipe de Avila Belbute Peres, Filippo Raso, Florent Bekerman, Foivos Tsimpourlas, Fotis Chantzis, Francis Song, Francis Zhang, Gaby Raila, Garrett McGrath, Gary Briggs, Gary Yang, Giambattista Parascandolo, Gildas Chabot, Grace Kim, Grace Zhao, Gregory Valiant, Guillaume Leclerc, Hadi Salman, Hanson Wang, Hao Sheng, Haoming Jiang, Haoyu Wang, Haozhun Jin, Harshit Sikchi, Heather Schmidt, Henry Aspegren, Honglin Chen, Huida Qiu, Hunter Lightman, Ian Covert, Ian Kivlichan, Ian Silber, Ian Sohl, Ibrahim Hammoud, Ignasi Clavera, Ikai Lan, Ilge Akkaya, Ilya Kostrikov, Irina Kofman, Isak Etinger, Ishaan Singal, Jackie Hehir, Jacob Huh, Jacqueline Pan, Jake Wilczynski, Jakub Pachocki, James Lee, James Quinn, Jamie Kiros, Janvi Kalra, Jasmyn Samaroo, Jason Wang, Jason Wolfe, Jay Chen, Jay Wang, Jean Harb, Jeffrey Han, Jeffrey Wang, Jennifer Zhao, Jeremy Chen, Jerene Yang, Jerry Tworek, Jesse Chand, Jessica Landon, Jessica Liang, Ji Lin, Jiancheng Liu, Jianfeng Wang, Jie Tang, Jihan Yin, Joanne Jang, Joel Morris, Joey Flynn, Johannes Ferstad, Johannes Heidecke, John Fishbein, John Hallman, Jonah Grant, Jonathan Chien, Jonathan Gordon, Jongsoo Park, Jordan Liss, Jos Kraaijeveld, Joseph Guay, Joseph Mo, Josh Lawson, Josh McGrath, Joshua Vendrow, Joy Jiao, Julian Lee, Julie Steele, Julie Wang, Junhua Mao, Kai Chen, Kai Hayashi, Kai Xiao, Kamyar Salahi, Kan Wu, Karan Sekhri, Karan Sharma, Karan Singhal, Karen Li, Kenny Nguyen, Keren Gu-Lemberg, Kevin King, Kevin Liu, Kevin Stone, Kevin Yu, Kristen Ying, Kristian Georgiev, Kristie Lim, Kushal Tirumala, Kyle Miller, Lama Ahmad, Larry Lv, Laura Clare, Laurance Fauconnet, Lauren Itow, Lauren Yang, Laurentia Romaniuk, Leah Anise, Lee Byron, Leher Pathak, Leon Maksin, Leyan Lo, Leyton Ho, Li Jing, Liang Wu, Liang Xiong, Lien Mamitsuka, Lin Yang, Lindsay McCallum, Lindsey Held, Liz Bourgeois, Logan Engstrom, Lorenz Kuhn, Louis Feuvrier, Lu Zhang, Lucas Switzer, Lukas Kondraciuk, Lukasz Kaiser, Manas Joglekar, Mandeep Singh, Mandip Shah, Manuka Stratta, Marcus Williams, Mark Chen, Mark Sun, Marselus Cayton, Martin Li, Marvin Zhang, Marwan Aljubeh, Matt Nichols, Matthew Haines, Max Schwarzer, Mayank Gupta, Meghan Shah, Melody Y. Guan, Melody Huang, Meng Dong, Mengqing Wang, Mia Glaese, Micah Carroll, Michael Lampe, Michael Malek, Michael Sharman, Michael Zhang, Michele Wang, Michelle Pokrass, Mihai Florian, Mikhail Pavlov, Miles Wang, Ming Chen, Mingxuan Wang, Minnia Feng, Mo Bavarian, Molly Lin, Moose Abdool, Mostafa Rohaninejad, Nacho Soto, Natalie Staudacher, Natan LaFontaine, Nathan Marwell, Nelson Liu, Nick Preston, Nick Turley, Nicklas Ansman, Nicole Blades, Nikil Pancha, Nikita Mikhaylin, Niko Felix, Nikunj Handa, Nishant Rai, Nitish Keskar, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Oona Gleeson,

Pamela Mishkin, Patryk Lesiewicz, Paul Baltescu, Pavel Belov, Peter Zhokhov, Philip Pronin, Phillip Guo, Phoebe Thacker, Qi Liu, Qiming Yuan, Qinghua Liu, Rachel Dias, Rachel Puckett, Rahul Arora, Ravi Teja Mullapudi, Raz Gaon, Reah Miyara, Rennie Song, Rishabh Aggarwal, RJ Marsan, Robel Yemiru, Robert Xiong, Rohan Kshirsagar, Rohan Nuttall, Roman Tsiupa, Ronen Eldan, Rose Wang, Roshan James, Roy Ziv, Rui Shu, Ruslan Nigmatullin, Saachi Jain, Saam Talaie, Sam Altman, Sam Arnesen, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Sarah Yoo, Savannah Heon, Scott Ethersmith, Sean Grove, Sean Taylor, Sebastien Bubeck, Sever Banesiu, Shaokyi Amdo, Shengjia Zhao, Sherwin Wu, Shibani Santurkar, Shiyu Zhao, Shraman Ray Chaudhuri, Shreyas Krishnaswamy, Shuaiqi, Xia, Shuyang Cheng, Shyamal Anadkat, Simón Posada Fishman, Simon Tobin, Siyuan Fu, Somay Jain, Song Mei, Sonya Egoian, Spencer Kim, Spug Golden, SQ Mah, Steph Lin, Stephen Imm, Steve Sharpe, Steve Yadlowsky, Sulman Choudhry, Sungwon Eum, Suvansh Sanjeev, Tabarak Khan, Tal Stramer, Tao Wang, Tao Xin, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Degry, Thomas Shadwell, Tianfu Fu, Tianshi Gao, Timur Garipov, Tina Sriskandarajah, Toki Sherbakov, Tomek Korbak, Tomer Kaftan, Tomo Hiratsuka, Tongzhou Wang, Tony Song, Tony Zhao, Troy Peterson, Val Kharitonov, Victoria Chernova, Vineet Kosaraju, Vishal Kuo, Vitchyr Pong, Vivek Verma, Vlad Petrov, Wanning Jiang, Weixing Zhang, Wenda Zhou, Wenlei Xie, Wenting Zhan, Wes McCabe, Will DePue, Will Ellsworth, Wulfie Bain, Wyatt Thompson, Xiangning Chen, Xiangyu Qi, Xin Xiang, Xinwei Shi, Yann Dubois, Yaodong Yu, Yara Khakbaz, Yifan Wu, Yilei Qian, Yin Tat Lee, Yinbo Chen, Yizhen Zhang, Yizhong Xiong, Yonglong Tian, Young Cha, Yu Bai, Yu Yang, Yuan Yuan, Yuanzhi Li, Yufeng Zhang, Yuguang Yang, Yujia Jin, Yun Jiang, Yunyun Wang, Yushi Wang, Yutian Liu, Zach Stubenvoll, Zehao Dou, Zheng Wu, and Zhigang Wang. Openai gpt-5 system card, 2026. URL https://arxiv.org/abs/2601.03267.

Aleksandrs Slivkins. Contextual bandits with similarity information. *Journal of Machine Learning Research*, 15(73):2533–2568, 2014. URL http://jmlr.org/papers/v15/slivkins14a.html.

Jialiang Wang, Shengxiang Xu, Hanmo Liu, Jiachuan Wang, Yuyu Luo, Shimin Di, Min-Ling Zhang, and Lei Chen. Learning to compose for cross-domain agentic workflow generation, 2026a. URL https://arxiv.org/abs/2602.11114.

Weijia Wang and Michèle Sebag. Multi-objective Monte-Carlo tree search. In Steven C. H. Hoi and Wray Buntine (eds.), *Proceedings of the Asian Conference on Machine Learning*, volume 25 of *Proceedings of Machine Learning Research*, pp. 507–522, Singapore Management University, Singapore, 04–06 Nov 2012. PMLR. URL https://proceedings.mlr.press/v25/wang12b.html.

Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. EvoAgentX: An automated framework for evolving agentic workflows. In Ivan Habernal, Peter Schulam, and Jörg Tiedemann (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 643–655, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-334-0. doi: 10.18653/v1/2025.emnlp-demos.47. URL https://aclanthology.org/2025.emnlp-demos.47/.

Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. Scoreflow: Mastering llm agent workflows via score-based preference optimization, 2025b. URL https://arxiv.org/abs/2502.04306.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. AgentDropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24013–24035, Vienna, Austria, July 2025c. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1170. URL https://aclanthology.org/2025.acl-long.1170/.

Zixu Wang, Bingbing Xu, Yige Yuan, Huawei Shen, and Xueqi Cheng. Do we always need query-level workflows? rethinking agentic workflow generation for multi-agent systems. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 5149–5165, San Diego, California,

United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.254. URL https://aclanthology.org/2026.findings-acl.254/.

Yangbo Wei, Zhen Huang, Lei He, Li Huang, Ting-Jung Lin, and Wei W. Xing. Vflow: Discovering optimal agentic workflows for verilog generation. In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 355–361, 2026. doi: 10.1109/ASP-DAC66049.2026.11420713. URL https://ieeexplore.ieee.org/document/11420713.

Bo Xiao, Haiyan Liu, Yitong Wang, Yanning Hou, Ke Xu, and Xingyi Zhang. Agentevo: cost-aware agentic workflow generation via adaptive multi-stage evolution. *Complex & Intelligent Systems*, 12, 2026. URL https://api.semanticscholar.org/CorpusID:288623541.

Shengxiang Xu, Jiayi Zhang, Shimin Di, Yuyu Luo, Liang Yao, Hanmo Liu, Jia Zhu, Fan Liu, and Min-Ling Zhang. Robustflow: Towards robust agentic workflow generation, 2025. URL https://arxiv.org/abs/2509.21834.

Xiao Xu and Ge Li. Chebyshev metric based multi-objective monte carlo tree search for combat simulations. In *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 607–612, 2017. doi: 10.1109/ICSTCC.2017.8107102. URL https://ieeexplore.ieee.org/document/8107102.

Zhenran Xu, Yiyu Wang, Yangxue, Longyue Wang, Weihua Luo, Kaifu Zhang, Baotian Hu, and Min Zhang. ComfyUI-r1: Exploring reasoning models for workflow generation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2999–3013, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.146. URL https://aclanthology.org/2026.findings-acl.146/.

Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/4a46fbfca3f1465a27b210f4bdfe6ab3-Paper.pdf.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Bo Yuan, Yun Zhou, Zhichao Xu, Kiran Ramnath, Aosong Feng, and Balasubramaniam Srinivasan. BayesFlow: A probability inference framework for meta-agent assisted workflow generation. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 3151–3179, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-386-9. doi: 10.18653/v1/2026.findings-eacl.165. URL https://aclanthology.org/2026.findings-eacl.165/.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly, 2025a. URL https://arxiv.org/abs/2502.07373.

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

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=LkzuPorQ5L.

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

Mingda Zhang, Wenjin Liu, Tiesunlong Shen, Qika Lin, Rui Mao, Erik Cambria, Xiaoying Tang, and Haoran Luo. Flowsteer: Towards agents designing agentic workflows via reinforced progressive canvas editing, 2026a. URL https://arxiv.org/abs/2602.01664.

Mingda Zhang, Tiesunlong Shen, Haoran Luo, Wenjin Liu, Zikai Xiao, Erik Cambria, and Xiaoying Tang. Skillflow: Flow-driven recursive skill evolution for agentic orchestration, 2026b. URL https://arxiv.org/abs/2605.14089.

Xiaoyuan Zhang, Xi Lin, Bo Xue, Yifan Chen, and Qingfu Zhang. Hypervolume maximization: A geometric view of pareto set learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=9ieV1hnuva.

Yuanshuo Zhang, Yuchen Hou, Bohan Tang, Shuo Chen, Muhan Zhang, Xiaowen Dong, and Siheng Chen. Gnns as predictors of agentic workflow performances, 2025e. URL https://arxiv.org/abs/2503.11301.

Mingming Zhao, Xiaokang Wei, Yuanqi Shao, Kaiwen Zhou, Lin Yang, Siwei Rao, Junhui Zhan, and Zhitang Chen. a^2flow : automating agentic workflow generation via self-adaptive abstraction operators, 2025. URL https://arxiv.org/abs/2511.20693.

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

APPENDIX

A Additional Results **18**
A.1 Accuracy-Cost Pareto Fronts 18
A.2 Task-Level versus Query-Level Search 18
A.3 Full Search Tree Visualizations 19
B Limitations **23**
C MoFlow Details **23**
C.1 Construction MOMDP and Prompt Templates 23
C.2 MoFlow Search Procedure 25
D Implementation Details **25**
E MoFlow Search Complexity Analysis **28**

A ADDITIONAL RESULTS

A.1 ACCURACY-COST PARETO FRONTS

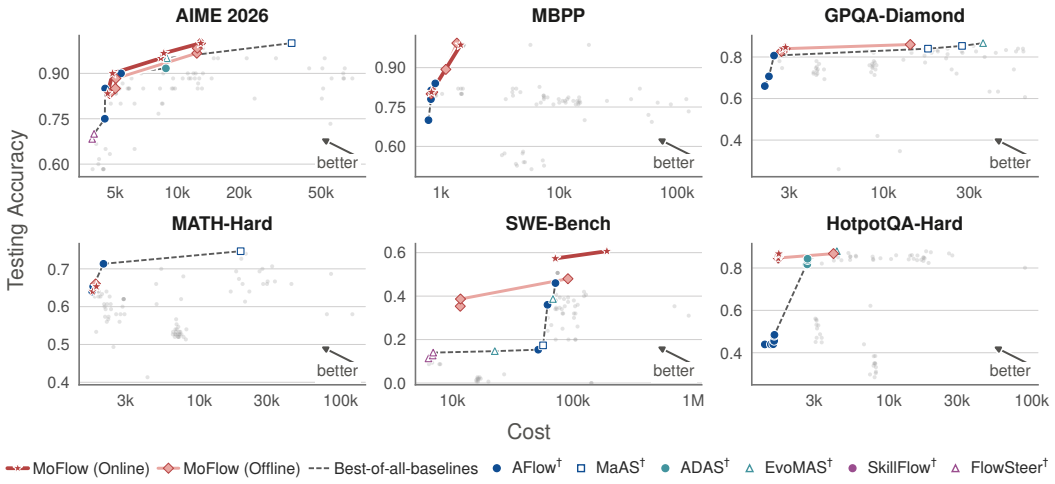


Figure 5: **Accuracy-cost Pareto fronts on the six benchmarks.** Test accuracy against tokens per problem under GPT-5 Mini (up and left is better). The dashed front combines all workflows generated by the six adapted baselines†, each rerun once per preference. Gray points are dominated workflows. **MoFlow, from a single search that never saw the eleven preferences, matches or extends this combined baseline front on five of the six benchmarks**, trailing only on MATH-Hard, where the baselines' most accurate workflow stays ahead at roughly ten times the cost.

Figure 5 plots each method's Pareto front on accuracy versus cost, the pair among our five objectives that practitioners care about most in deployment.

A.2 TASK-LEVEL VERSUS QUERY-LEVEL SEARCH

MoFlow runs at one of two granularities. *Task-level* search builds one tree per benchmark on its build set and serves every testing preference from that tree by lookup. *Query-level* search builds a small front for each incoming query, scoring candidates against the query's paraphrases and repeated runs. Appendix E derives the complexity of both modes. Figure 6 compares the best value each mode reaches on every objective, and Table 5 reports build cost, per-problem inference cost, and generalization. We measure generalization by the *hypervolume generalization ratio*

$$\rho_{\mathrm{HV}} = \frac{\mathrm{HV}_{\text{held-out}}}{\mathrm{HV}_{\text{in-sample}}}, \tag{4}$$

where $\mathrm{HV}_{\text{in-sample}}$ is a tree's hypervolume on the inputs its search evaluated and $\mathrm{HV}_{\text{held-out}}$ its hypervolume on unseen inputs. For a task-level tree these are the build and held-out sets of §4. For

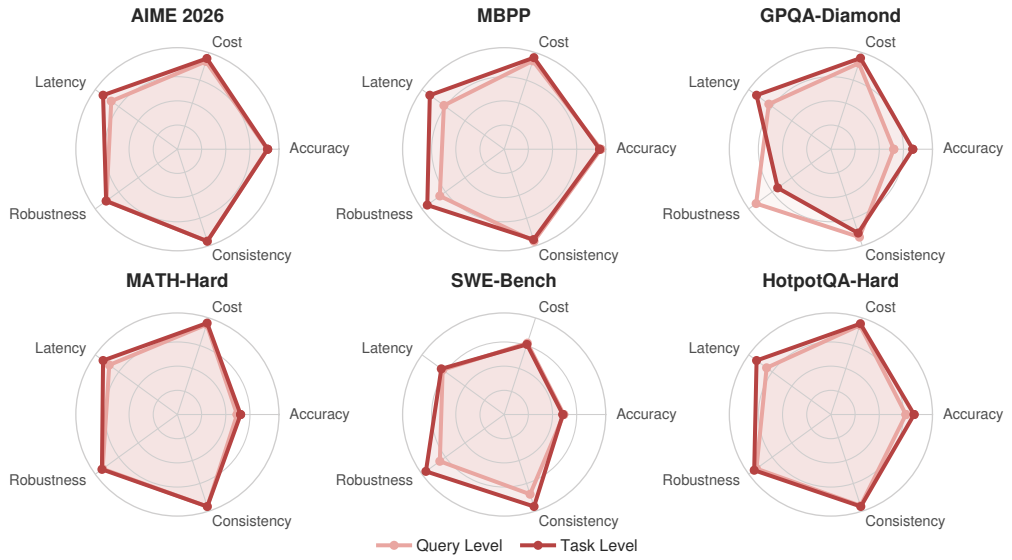


Figure 6: **Per-objective performance of task-level versus query-level search.** One tree per benchmark (*task-level*) against one small tree per query (*query-level*), under the same setting as §4. Each axis shows the best value a search mode attains across its eleven served workflows, with cost and latency inverted so that outward is always better. **Task-level search matches or exceeds query-level search on nearly all objectives across the six benchmarks.**

Table 5: **Cost comparison between task-level and query-level search.** Search cost is the execution tokens spent building each tree. Inference cost and latency are the mean tokens and sequential calls per problem, averaged over the workflows returned for the eleven preferences. ρ_{HV} is the hypervolume generalization ratio (Eq. 4). Query-level trees are cheaper to build, but the workflows they return spend up to 11.6× the tokens and 6.6× the calls per problem at inference. **Query-level search overfits the inputs it was built on: ρ_{HV} falls as low as 0.28 on unseen draws of the same query, against at least 0.66 for every task-level tree.**

	Search cost (M tokens)		Inference cost (tokens)		Inference latency (calls)		ρ_{HV} (↑)	
Benchmark	Task	Query	Task	Query	Task	Query	Task	Query
AIME 2026	132.3	111.9	5.6k	23.0k	1.36	4.27	0.68	0.37
MATH-Hard	115.7	74.1	5.7k	12.5k	1.00	3.80	0.93	0.73
MBPP	134.9	85.4	2.1k	20.1k	1.19	7.89	0.87	0.28
SWE-Bench	597.2	452.1	102.7k	115.3k	6.53	7.36	0.66	0.63
GPQA-Diamond	232.0	162.7	2.8k	32.5k	1.00	6.25	1.95	0.93
HotpotQA-Hard	122.8	92.2	5.6k	15.8k	1.36	5.59	0.86	0.65

a query-level tree, the in-sample inputs are the paraphrases and repeated runs scored during search, the held-out inputs are unseen paraphrases, and Table 5 reports the ratio of mean hypervolumes across a benchmark's query trees. ρ_{HV} near one means the front generalizes, and a value well below one means the search overfit. Task-level GPQA-Diamond ($\rho_{\mathrm{HV}} = 1.95$) exceeds one because its 20-question build split is harder than the 50-question held-out split (the same generated workflows score 0.63 accuracy in-sample but 0.83 held out), and hypervolume compounds this per-objective gap multiplicatively.

A.3 Full Search Tree Visualizations

Figures 7–9 show three query-level trees in full, one per task type: code repair (SWE-Bench), mathematics (MATH-Hard), and scientific QA (GPQA-Diamond). We visualize query-level rather than task-level trees because the latter, at roughly 240–490 decision nodes, are too large to render completely.

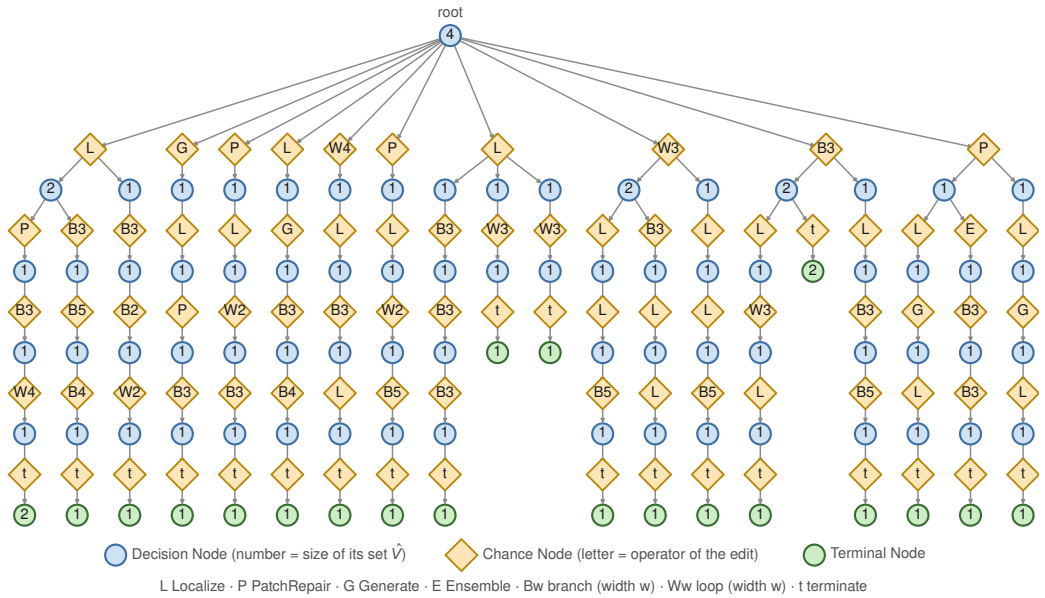


Figure 7: **A complete query-level search tree on SWE-Bench,** built with GPT-5 Mini. The tree contains 90 decision nodes (blue), 83 chance nodes (gold), and 20 evaluated terminals (green). The number in each circle is the size of the node's CCS $\hat{V}(s)$: a 1 means the node holds a single trade-off. Two terminals show a 2 because one of their three repeated runs solved the problem while another failed but used fewer tokens or calls, so neither run dominates. Each terminal stores these pruned per-run returns in $\hat{V}(s)$, while Eq. 1 backs up their average $\mathbb{E}[R]$. The tree thus records that a workflow can fail and fail cheaply.

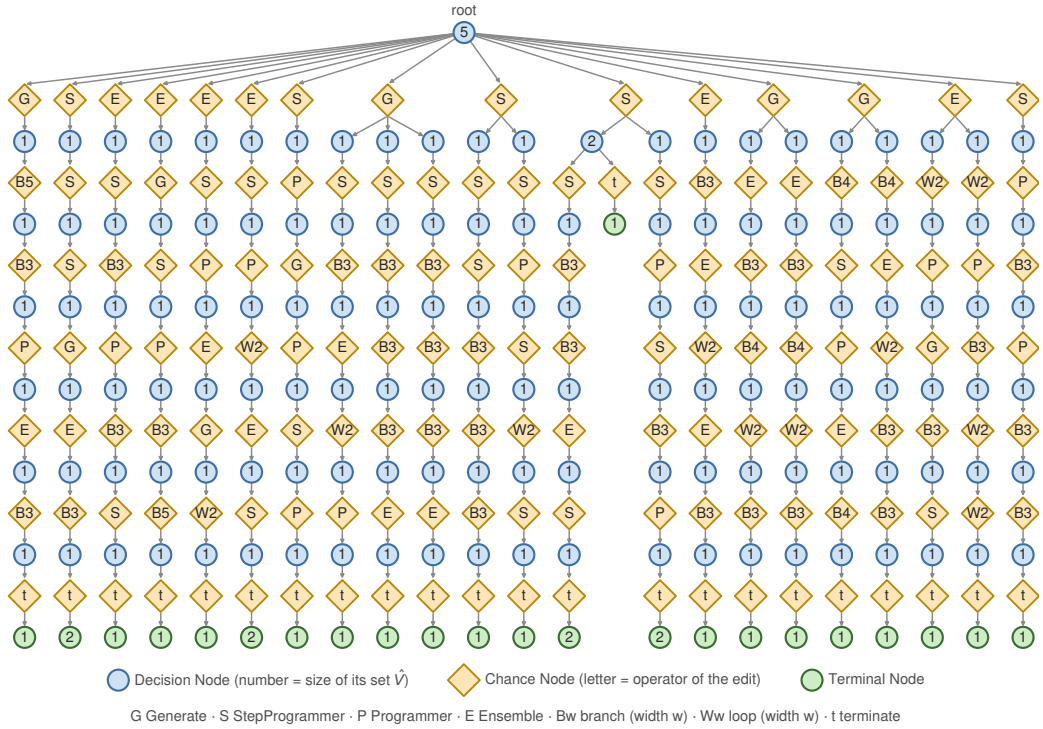


Figure 8: **A complete query-level search tree on MATH-Hard.** It contains 156 decision nodes, 148 chance nodes, and 23 evaluated terminals. The root keeps five trade-offs, ranging from an accurate workflow at 10.8k tokens down to a single-call workflow at 1.0k tokens that fails, and the blue "2" at depth 1 shows that an interior node can also keep more than one trade-off. The search favors code execution, selecting `StepProgrammer` or `Programmer` in 45 of its 148 actions.

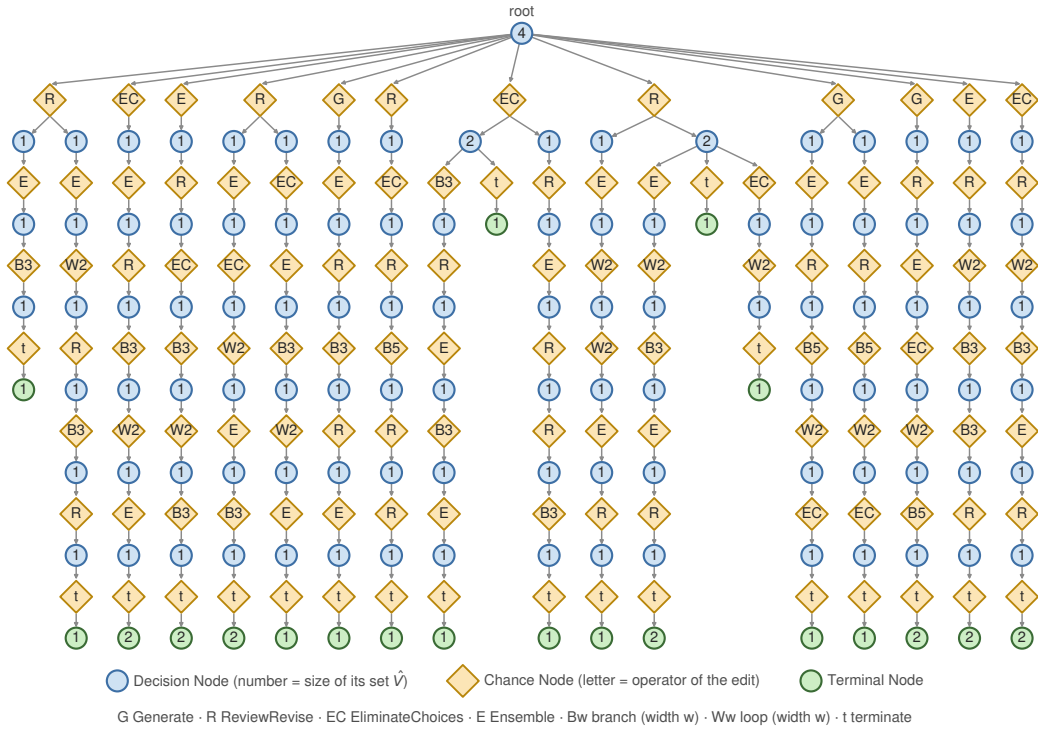


Figure 9: **A complete query-level search tree on GPQA-Diamond.** The tree contains 122 decision nodes, 116 chance nodes, and 20 evaluated terminals. The root spans the widest cost range of the three trees: its accurate workflow costs 82k tokens and 14 calls, while its two-call workflows stay under 9k tokens. Seven of the 20 terminals show a 2, each keeping both its cheapest accurate run and its cheapest failed run. The tree thus records that the same workflow can succeed, or fail at a slightly lower cost. The search is critique-driven, selecting the `ReviewRevise` operator in 42 of its 116 actions.

B LIMITATIONS

We note three limitations of MoFlow. First, it compares value vectors through the linear score $\boldsymbol{w}^\top \boldsymbol{v}$, so it covers the convex hull of the Pareto front but not its concave regions, which only nonlinear scalarizations (e.g., Chebyshev) can reach. Second, like AFlow (Zhang et al., 2025d), it requires a manually designed operator pool for each task. With a large open pool, the trial budget alone could not substitute for a stronger proposer. Third, to save cost, every online search stops early once the hypervolume of the executed front plateaus (Appendix D), a rule that halted all reported runs between trials 40 and 50, so we never observe MoFlow (online) under a substantially larger budget. Moreover, the fixed operator pool and horizon bound the structurally distinct workflows the search can reach (even though free-form roles leave the raw workflow set unbounded), so additional trials would likely revisit the front rather than extend it.

C MOFLOW DETAILS

Appendix C.1 gives the state and action spaces of the MOMDP together with the prompt templates, and Appendix C.2 the full search procedure.

C.1 CONSTRUCTION MOMDP AND PROMPT TEMPLATES

State representation. A state is an immutable `WorkflowGraph`. Two partial workflows with the same structure share a canonical key and map to the same node, so a workflow reachable by different edit sequences is visited only once.

Action space. An action is one of three atomic edits.

- `AddOperator(`*o*`, width, role)` appends operator *o* from the task's operator pool (below). The `role` is a free-text label that names the node's prompt archetype (e.g., "step-by-step solver" or "code verifier").
- `AddControl(branch|loop, width)` wraps an operator in a control structure: `branch` runs `width` parallel attempts and keeps the best, while `loop` refines sequentially for `width` rounds.
- `Terminate` closes the workflow.

Roles are not edited independently. The proposer assigns an initial role when it proposes each node-adding edit, and the realizer rewrites that role into up to N_r distinct specializations when the edit is instantiated (realizer prompt below). An edit is *legal* only if it yields a valid workflow: a terminal workflow cannot be extended, `Terminate` requires a non-empty graph whose last operator emits an answer (e.g., `Generate` or `Ensemble`), widths are bounded, and depth cannot exceed the horizon H. These constraints define the legal action set $A(s)$.

Operator pools. Below we show the operators we defined for each benchmark.

- **AIME 2026, MATH-Hard**: `Generate`, `Ensemble`, `Programmer`, `StepProgrammer`;
- **MBPP**: `Generate`, `Ensemble`, `ReviewRevise`, `TestCode`;
- **SWE-Bench**: `Generate`, `Localize`, `PatchRepair`, `Ensemble`, `Custom`;
- **GPQA-Diamond**: `Generate`, `Ensemble`, `ReviewRevise`, `EliminateChoices`;
- **HotpotQA-Hard**: `Generate`, `Ensemble`, `Custom`, `Decompose`, `EvidenceSelect`, `GroundCheck`.

We purposefully design operators aligned with those of existing baselines to ensure a fair comparison.

Prompt templates. Three templates drive the proposer, the realizer, and the executor, and all three are fixed across the four models and six benchmarks. The **proposer** template returns candidate actions at a decision node.

Proposer Prompt

```
You are designing an LLM workflow (a DAG of operators) to solve the
following task:
{task}

The current partial workflow has these operators in order: {ops}.

Available operators (choose only from these, by exact name):
{catalogue}

Propose up to {n} good NEXT atomic edits to extend or finish the
workflow. An edit is one of: a base operator; "loop" (wrap an
operator to refine it sequentially for `width` passes); "branch"
(run `width` parallel attempts of an operator and keep the best);
or "Terminate".

Pick operators and roles that fit THIS task; the last operator
before Terminate must produce the task's required answer/output
(do not finish on an analysis-only step).

Reply with ONLY a JSON list, each item {"op":
<operator|"loop"|"branch"|"Terminate">, "operator": <inner
operator for loop/branch, else omit>, "width": <int >= 1>, "role":
<short label>, "why": <one-line rationale>}.
```

The **realizer** template turns one selected action into N_r distinct successors.

Realizer Prompt

```
You are instantiating one step of an LLM workflow for the following
task:
{task}

The workflow so far has these operators in order: {ops}.

The next step adds a `{operator}` operator (width {width}). That
operator's job: {operator_desc}

Propose {n} DISTINCT concrete ways to play that step -- each a
different role/approach the operator could take FOR THIS TASK (a
distinct prompt archetype, reasoning strategy, or specialization),
so that executing them could plausibly lead to different outcomes.
Each role must fit the operator's job and this task.

Reply with ONLY a JSON list of {n} items, each {"role": <short
distinct approach label, <= 40 chars>, "why": <one-line
rationale>}.
```

The **executor** assembles each operator call from four parts in order: the benchmark's *task description*, the problem behind an `Input:` header, the *operator-specific instruction*, and the benchmark's *answer format*. Only the task description and answer format depend on the benchmarks, while the operator instruction is fixed and the realizer fills in its {role} slot. The example below shows the assembled prompt for a `Generate` step on MATH-Hard.

Executor Prompt (`Generate` on MATH-Hard)

```
You are an expert competition mathematician. Solve the problem
carefully, reasoning step by step.

Input:
{problem}
```

```
Role: {role}. Reason step by step, then give your best answer.

Put your final answer in \boxed{...} on the last line. Give it in
exact, simplest form (reduced fraction, exact radical, or integer),
not a decimal approximation.
```

C.2 MoFlow Search Procedure

Set operations. The backups of Eqs. 1 and 2 use three set operations. *Prune* discards every vector dominated by another in the set. *Union* merges two sets and then prunes. The Minkowski sum $X \oplus Z = \text{prune}\{x + z : x \in X, z \in Z\}$ adds an immediate reward set to a future value set. Because prune removes only Pareto-dominated vectors, each node stores the full Pareto front, a superset of the minimal CCS.

Optimistic chance backup. The union over successors in Eq. 1 is deliberately optimistic rather than exact. The exact CHVI backup (Barrett & Narayanan, 2008; Painter et al., 2020) is the probability-weighted Minkowski sum over successors. It values an action by averaging over the realizations of the edit. Our union instead keeps every vector achievable under at least one realization. When a preference $\boldsymbol{w}$ takes the scalarized maximum of this set, the action is credited with its best realization, as though the search could choose the outcome. In reality, the search only observes which realization occurs, so the union upper-bounds the exact value for every $\boldsymbol{w}$. We accept the optimistic bias for two reasons. The union is much cheaper, since the exact sum multiplies front sizes across successors while the union only merges them (Appendix E). More importantly, the bias affects only which workflow is generated, while every reported number is an executed return.

CZT metric. CZT covers the context–arm space $\mathcal{P}_s = \Delta^{D-1} \times A(s)$ with balls under the metric $d_s((\boldsymbol{w}, a), (\boldsymbol{w}', a')) = \|\boldsymbol{w} - \boldsymbol{w}'\|_\infty$ if $a = a'$ and U otherwise, where $V_{\max}$ is the largest possible scalarized return and $U \geq V_{\max}$ keeps distinct actions incomparable (Painter et al., 2020). Each arm is identified by its edit's property tuple (operator, width, and role for `AddOperator`, or control kind, width, and wrapped operator for `AddControl`), so a re-proposed edit joins an existing ball only when the tuples match.

Selection and zooming. On each visit with context $\boldsymbol{w}$, CZT scores the relevant balls by the index of Eq. 3. A ball is relevant if its action was proposed on this visit and $\boldsymbol{w}$ lies in the ball's domain, the portion of B not covered by a finer same-action ball. Because distinct actions sit at distance $U \geq V_{\max}$, only same-action balls can tighten the min in Eq. 3, so a well-explored neighbor constrains the optimism of an under-explored ball. The visit applies the action of the largest-index ball and folds the observed scalarized reward into $\nu(B)$. Once $\text{conf}(B) \leq r(B)$, where $\text{conf}(B) = \kappa\sqrt{\log N/(1 + n(B))}$ for a ball selected $n(B)$ times under trial budget N, the ball activates a half-radius child of the same action centered at $\boldsymbol{w}$.

D Implementation Details

Table 6 lists all MoFlow hyperparameters and reproducibility details. The rest of this section describes the motivation behind certain design and hyperparameter choices.

Dataset and evaluation. We split each benchmark into a build set and a held-out set. Each problem carries $m = 3$ paraphrases, generated by GPT-5 Mini at temperature 0.9 with a prompt that varies only wording so the gold answer is unchanged. Grading uses the boxed answer on the two math suites, reference tests on MBPP, the official harness on SWE-Bench, the gold letter on GPQA-Diamond, and token-level F1 on HotpotQA-Hard. All methods share this grading, and a single trial measures all five objectives: consistency comes from the variance of accuracy across the J repeated runs, and robustness from its variance across the m paraphrases. The factor 4 in $\text{clip}(1 - 4\text{Var}(\text{acc}), 0, 1)$ is not tuned. Per-problem accuracy lies in $[0, 1]$, so its variance is at most $1/4$ and the factor simply rescales the score to $[0, 1]$. Every LLM call is routed through the LiteLLM proxy server (BerriAI, 2023), and cost sums the per-call token usage the proxy meters. Latency

counts sequential LLM calls rather than wall-clock time, which varies with provider, hardware, and load, so parallel calls within an `Ensemble` or `branch` count as one round.

Reference point. Hypervolume is meaningful only when every served workflow dominates the reference point $\boldsymbol{r}_0$, because a workflow that falls outside it contributes zero volume and can distort the method ranking. The most expensive workflow served in our experiments uses 1.9×10^5 tokens per problem and the slowest makes 12.5 sequential calls, both on SWE-Bench. We set the cost and latency coordinates of $\boldsymbol{r}_0$ to 3×10^5 tokens and 20 calls, leaving ample headroom beyond these observed extremes.

Baseline adaptation. We adapt each baseline by changing only the score it optimizes, replacing its scalar objective with $\boldsymbol{w}^\top \boldsymbol{v}$ and rerunning the method once per testing preference. All other aspects of the baseline (design, implementation, prompts, and hyperparameters) remain untouched. Every method searches on the same build set, and every workflow is scored under the shared protocol ($J = 3$ repeated runs and $m = 3$ paraphrases), so each objective is measured identically across methods.

Uncertainty-gated objective evaluation. Accuracy, robustness, and consistency can only be measured from a finished workflow. At a partial workflow, f_θ instead predicts these three objectives along with its own uncertainty. We execute only the nodes where that uncertainty exceeds the gate, which in practice triggers at the leaves of a small fraction γ of trials. At a gated node, we let a completion policy π_c fill the open slots of s until termination, execute a diverse beam of k completions, and replace the prediction with their non-dominated vectors,

$$\hat{V}_{\mathrm{exec}}(s) \approx \mathrm{prune}\big(\{R(\mathrm{exec}(c)) : c \in \mathrm{Beam}_k(\pi_c, s)\}\big). \tag{5}$$

The executed returns also feed the replay buffer on which f_θ is trained, so even when f_θ is confidently wrong at some node and the gate never fires there, the finished workflows that extend it are still executed and retraining on their results corrects the prediction.

Total cost. Each baseline rerun receives the method's default budget (5.6–28.7M execution tokens per preference, depending on method and benchmark), so a baseline spends roughly 62–316M tokens per benchmark across the eleven preferences. A single online MoFlow search uses 116–597M tokens, the same order as one baseline sweep, while offline MoFlow serves all eleven preferences from just 2.9–3.6M proposer and realizer tokens. Across the full study, the MoFlow searches behind Table 2, Table 4, Table 3, and Table 5 consume 3.1B metered tokens over about 450,000 execution calls. Including the estimated 9B tokens for baseline reruns, the study totals roughly 12B tokens, about $14,700 at September 2026 on-demand rates.

Table 6: **MoFlow hyperparameters and reproducibility information.** All settings are shared across the six benchmarks and four base models except the horizon and set sizes, noted in their rows.

Name	Value
Construction MOMDP and search	
Number of objectives D	5
Horizon H	6 (4 on SWE-Bench)
Trial cap N (online)	100; from trial 40, stop once 10-trial HV gain $< 1\%$
Trial budget N (offline)	300, no early stop
Action proposals per visit b	4
Realizations per action N_r	3
Completion-beam width k	3
CZT confidence constant κ	4
Random seed	0
Objectives and evaluation	
Repeated runs J	3 per problem
Paraphrases m	3
Execution temperature	0.7
Reference point $\boldsymbol{r}_0$	$[0,\ -3{\times}10^5,\ -20,\ 0,\ 0]$
Testing preferences	11 fixed $\boldsymbol{w}$
Build / held-out set sizes	$10/20$ on AIME 2026, $20/50$ elsewhere
Value model f_θ	
Architecture	2-layer message-passing GNN, width 64, role dim 8
Role encoder	all-MiniLM-L6-v2
Predicted objectives	accuracy, robustness, consistency
Uncertainty gate	0.15
Online refit	every 32 replay entries; Adam, lr 10^{-2}, 60 epochs
Offline training	Adam, lr 10^{-3}, decay 10^{-4}, 600 epochs, patience 100
Offline training pool	801 executed workflows from six online searches
Offline train / val split	$70\%/30\%$ by workflow hash ($558/243$ workflows)
Compute	
Execution tokens (online)	116–597M per search
LLM tokens (offline)	2.9–3.6M per search
Total study cost	$\sim$12B tokens, $\sim$\$14,700

E MoFlow Search Complexity Analysis

MoFlow searches at two granularities, defined and compared in Appendix A.2. Task-level search pays for one tree per task and serves every later preference from it, while query-level search pays for a new small front on each incoming query. This section derives the cost of a single trial and extends it to both granularities.

Cost of one trial. The *executed front* is the non-dominated set of return vectors from all workflows executed so far in the search. Let N be the number of trials, H the horizon, b the per-visit proposal budget, N_r the realizations per selected action, D the number of objectives, k the completion-beam width, F the maximum per-node front size after pruning, $|\pi|$ the number of tree nodes in the policy unrolled for one preference, Φ the maximum number of vectors on the executed front during the search, β the maximum number of active CZT balls at a decision node, E the cost of one workflow evaluation, and $\ell \ll E$ the cost of one proposer or realizer LLM call. Because the proposer redraws at most b actions on every visit, a decision node's branching factor is not b but the number $\bar{A} \le \min\{|A(s)|, bT_s\}$ of distinct actions ever registered there, where T_s is the number of visits to node s. The rollout, workflow execution, set backups, executed-front update, and CZT selection sum to

$$\underbrace{O\big((1+\gamma k)E + H\ell\big)}_{\text{executions and LLM calls}} + \underbrace{O\big(H(\bar{A}+N_r)^2F^2D\big)}_{\text{set backups}} + \underbrace{O\big((1+\gamma k)\Phi D\big)}_{\text{executed-front update}} + \underbrace{O\big(H\beta^2 D\big)}_{\text{CZT selection}}, \tag{6}$$

with γ the fraction of trials whose rollout ends at an uncertain interior node and so triggers a completion beam. The beam's completion policy adds $O(kH)$ proposer calls of its own on those trials, absorbed into $O(H\ell)$ since k is a constant. The first term dominates, since executing workflows costs far more than pruning small vector sets, except in the offline mode, which executes no workflows while building the tree and incurs only the $O(H\ell)$ proposer and realizer calls.

Derivation of Eq. 6. The four terms follow one trial in the order its costs are paid. The rollout visits at most H decision nodes and makes about two LLM calls at each. One proposer call redraws candidate actions and one realizer call samples the chosen action's outcome, so the rollout pays $O(H\ell)$. The execution cost then depends on how the trial ends. A terminal leaf is executed once at cost E, and a confident interior leaf is valued by f_θ with no execution. An uncertain interior leaf triggers a completion beam, which executes its k completions at cost kE. A trial therefore executes workflows at most $1 + \gamma k$ times. Combined with the rollout's LLM calls, this yields the first term, $O\big((1+\gamma k)E+H\ell\big)$. After execution, the set backups run once at every level of the visited path, and their cost comes from pruning. Discarding Pareto-dominated vectors from n candidates in $\mathbb{R}^D$ takes $O(n^2D)$ (one $O(D)$ dominance check per pair) (Roijers et al., 2013). The chance backup of Eq. 1 unions the fronts of at most N_r realized successors, each of size at most F, so it prunes at most N_rF candidates at cost $O(N_r^2F^2D)$. The decision backup of Eq. 2 likewise unions the $\bar{A}$ registered actions' fronts, so it prunes at most $\bar{A}F$ candidates at cost $O(\bar{A}^2F^2D)$. Since $\bar{A}^2 + N_r^2 \le (\bar{A}+N_r)^2 \le 2(\bar{A}^2+N_r^2)$, summing the two backups over the at most H levels gives the second term, $O\big(H(\bar{A}+N_r)^2F^2D\big)$. The trial's $O(1+\gamma k)$ executed return vectors then enter the executed front. Inserting one vector runs one $O(D)$ dominance check against each of the at most Φ stored vectors, so the executed-front update gives the third term, $O\big((1+\gamma k)\Phi D\big)$. Finally, choosing an action at each visited node evaluates the active CZT balls using the index of Eq. 3. Each index takes a minimum over the at most β active balls, and each pair of balls requires one $O(D)$ distance evaluation. One selection therefore costs $O(\beta^2 D)$, and the at most H selections give the fourth term, $O\big(H\beta^2D\big)$.

Task-level search. We run N trials once and store the resulting executed front and tree, at N times Eq. 6,

$$O\big(N(1+\gamma k)E + NH\ell\big) + O\big(NH(\bar{A}+N_r)^2F^2D\big) + O\big(N(1+\gamma k)\Phi D\big) + O\big(NH\beta^2D\big).$$

When actions have several realizations, inference unrolls the conditional policy by scanning, at each of its $|\pi|$ nodes, the $\bar{A}$ registered actions' fronts of size at most F, at $O(|\pi|\bar{A}FD)$. When realizations are unique, inference is a single scan of the executed front, at $O(\Phi D)$.

Query-level search. The system receives a query q and runs a smaller search for it. A single query gives only a binary success signal, so we evaluate workflows on a small neighborhood of paraphrases and repeated samples. Accuracy is then the mean success rate on the original query, robustness is stability across paraphrases, and consistency is stability across repeated runs. The cost is Eq. 6 with N_q in place of N,

$$O\big(N_q(1+\gamma_q k)E_q + N_q H\ell\big) + O\big(N_q H(\bar{A}+N_r)^2F^2D\big) + O\big(N_q(1+\gamma_q k)\Phi_q D\big) + O\big(N_q H\beta^2 D\big),$$

where E_q is one evaluation over the query neighborhood, Φ_q the maximum number of vectors on that query's executed front during its search, and γ_q the gated fraction of the query's trials. Unlike in task-level search, the front itself is the answer. Across many queries the replay buffer for f_θ is shared, so later queries may need fewer completion beams and fewer trials.

Comparison with the baselines. Every baseline spends its entire budget on a single trade-off, implicit or fixed, so a new preference $\boldsymbol{w}$ forces either a fresh search (AFlow, ADAS, EvoMAS) or a fresh training run (MaAS, FlowSteer, SkillFlow). Covering a set $\mathcal{W}$ of testing preferences therefore multiplies each method's full budget by $|\mathcal{W}|$, e.g. $O(|\mathcal{W}|NE)$ for the search-based generators. MoFlow instead pays $O\big(N((1+\gamma k)E + H\ell)\big)$ once in task-level search, and only its $O(NH\ell)$ proposer and realizer portion in the offline mode. Every later preference is then answered with zero LLM calls, either by unrolling the conditional policy at $O(|\pi|\bar{A}FD)$ or, when realizations are unique, by a single scan of the executed front at $O(\Phi D)$.