On Time, Within Budget: Constraint-Driven Online Resource Allocation for Agentic Workflows

Xinglin Wang[1]*, **Zishen Liu**[1]*, **Shaoxiong Feng**[2]†, **Peiwen Yuan**[1], **Yiwei Li**[1], **Jiayi Shi**[1], **Yueqi Zhang**[1], **Chuyi Tan**[1], **Ji Zhang**[1], **Boyuan Pan**[2], **Yao Hu**[2], **Kan Li**[1]†
[1] School of Computer Science, Beijing Institute of Technology
[2] Xiaohongshu Inc
{wangxinglin,liuzishen,peiwenyuan,liyiwei}@bit.edu.cn
{zhangyq,shijiayi,tanchuyi,likan}@bit.edu.cn
shaoxiongfeng2023@gmail.com {panboyuan,xiahou}@xiaohongshu.com

Abstract

Agentic systems increasingly solve complex user requests by executing orchestrated workflows, where subtasks are assigned to specialized models or tools and coordinated according to their dependencies. While recent work improves agent efficiency by optimizing the performance–cost–latency frontier, real deployments often impose concrete requirements: a workflow must be completed within a specified budget and before a specified deadline. This shifts the goal from average efficiency optimization to maximizing the probability that the entire workflow completes successfully under explicit budget and deadline constraints. We study *constraint-driven online resource allocation for agentic workflows*. Given a dependency-structured workflow and estimates of success rates and generation lengths for each subtask–model pair, the executor dynamically allocates models and parallel samples across simultaneously executable subtasks while managing the remaining budget and time. We formulate this setting as a finite-horizon stochastic online allocation problem and propose *Monte Carlo Portfolio Planning* (MCPP), a lightweight closed-loop planner that directly estimates constrained completion probability through simulated workflow executions and replans after observed outcomes. Experiments on CodeFlow and ProofFlow demonstrate that MCPP consistently improves constrained completion probability over strong baselines across a wide range of budget–deadline constraints[1].

1 Introduction

Recent advances in large language models (LLMs) have substantially expanded the capabilities of agentic systems, enabling planning and acting in interactive environments [Yao et al., 2022, Liu et al., 2024, Erdogan et al., 2025, Zheng et al., 2025], tool use [Schick et al., 2023, Qin et al., 2024, Patil et al., 2024, Ahn et al., 2026, Xiao et al., 2026], and complex multi-step task execution [Mialon et al., 2024, Jimenez et al., 2024, Zhang et al., 2026d]. To handle open-ended and long-horizon objectives, recent agentic systems increasingly rely on workflow orchestration to expose parallelism, exploit specialization, and manage long-range dependencies: a user request is decomposed into subtasks, assigned to specialized agents, models, or tools, and executed concurrently when dependencies allow [Zhuge et al., 2024, Zhang et al., 2025d, Team et al., 2026]. This shift turns agent execution from an isolated model response into a graph-structured workflow, making the organization of subtasks and

*Equal contribution.
†Corresponding author.
[1] Our code and data have been released on https://github.com/WangXinglin/MCPP.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

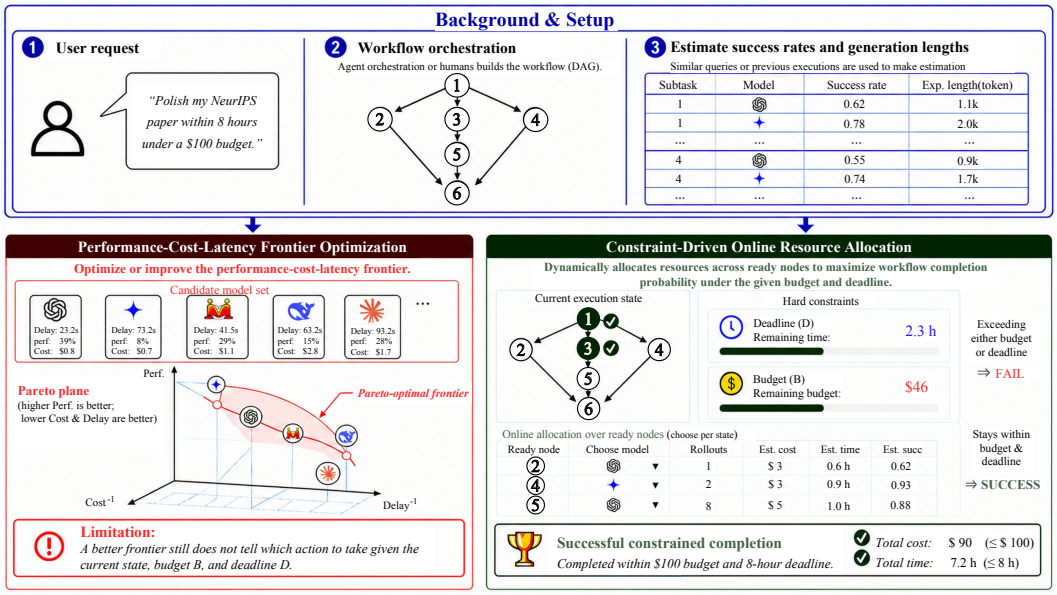

Figure 1: Comparison between performance–cost–latency frontier optimization and constraint-driven online resource allocation. A user request is first transformed into a static workflow, and success-rate and generation-length estimates for subtask–model pairs can be obtained from similar past queries or executions. Previous methods optimize or improve the performance–cost–latency frontier, but a better frontier still does not specify which action to take given the current workflow state, budget B, and deadline D. In contrast, our method dynamically allocates models and parallel rollouts to ready subtasks during execution to maximize the probability that the workflow completes successfully within the specified budget and deadline.

their information flow an optimizable object in agentic systems [Zhang et al., 2025a,b,c, Wang et al., 2025e,f].

Yet a well-designed workflow does not by itself determine how the executor should carry it out once execution begins, including which model to invoke for each currently executable subtask and how many parallel samples to draw [Laju et al., 2026]. Existing work typically studies these execution-time choices from an efficiency-oriented perspective, adapting model selection, routing decisions, or inference behavior to improve the performance–cost–latency frontier [Wang et al., 2025a, Zhang et al., 2026b, Ma et al., 2026, Fan et al., 2026]. However, users in real deployments rarely ask whether an agent is generally faster, cheaper, or more accurate on average. They ask whether this particular workflow can be completed within a specified budget B and before a specified deadline D (e.g., polishing a NeurIPS submission within eight hours under a $100 budget). A better performance–cost–latency frontier still does not answer this instance-specific question, as it does not specify how to spend the remaining budget and time at each workflow state to maximize the chance of finishing within B and D. Under such requirements, cost and time become part of the success condition, shifting the objective from improving performance–cost–latency frontiers to maximizing the probability that the entire workflow completes successfully within the specified budget and deadline (Figure 1).

Motivated by this gap, we study *constraint-driven online resource allocation for agentic workflows*. The workflow itself may be generated by an orchestrator, designed by humans, or optimized by an external workflow search method. Our focus begins once this workflow is given and execution starts. We assume that, for each subtask–model pair, the executor has estimates of single-attempt success probability and expected generation length[2], which can be obtained from historical executions, offline calibration, or learned prediction methods [Pacchiardi et al., 2024, Pan et al., 2025, Ding et al., 2025, Zhang et al., 2026b, Xie et al., 2026].

[2]Estimating these quantities is complementary to our goal.

Given these estimates, the remaining challenge is to decide how to allocate resources during execution. This is a non-trivial online problem: each allocation changes the probability of completing currently ready subtasks, consumes budget and wall-clock time, and affects which downstream subtasks may become available later. A static assignment of models or sampling widths is therefore insufficient, since the best decision depends on the evolving workflow state, the remaining budget, the remaining time, and the stochastic success or failure events observed so far. We formalize the objective as maximizing the *constrained completion probability*,

$$\Pr_{\pi}(T_{\text{solve}} \leq D,\ C_{\pi} \leq B)\,,$$

where T_{solve} is the workflow completion time and C_{π} is the total execution cost under policy π. Although exact dynamic programming gives a clean conceptual solution, it quickly becomes intractable as the completed-subtask set, allocation choices over currently executable subtasks, and stochastic execution outcomes grow combinatorially. To obtain a practical solver, we propose Monte Carlo Portfolio Planning (MCPP), a simple yet effective online planning framework. At each execution state, the planner evaluates candidate resource-allocation actions through simulated workflow executions, selects the action with the highest estimated constrained completion probability, observes the actual outcomes, and replans from the updated state. This directly targets the user-facing success event instead of optimizing a soft performance–cost–latency proxy.

To validate the effectiveness of our approach, we conduct experiments on CodeFlow [Wang et al., 2025c] and ProofFlow [Cabral et al., 2025], two representative benchmarks for dependency-structured code and proof tasks. Across varying budget–deadline constraints, our method improves constrained completion probability over strong baselines. We further conduct noise-injection experiments to evaluate the robustness of our method under noisy estimates of subtask success rates and generation lengths.

Our contributions are summarized as follows:

- We introduce *constraint-driven online resource allocation for agentic workflows* to capture a common deployment requirement: whether a given workflow can be completed successfully within a specified budget and deadline. In this setting, budget and deadline are treated as hard success conditions, and the objective is to maximize constrained completion probability.
- We formulate this setting as a finite-horizon stochastic online allocation problem and propose *Monte Carlo Portfolio Planning* (MCPP), a closed-loop planner with a safe-improvement guarantee that directly estimates constrained completion probability to allocate models and parallel samples online.
- We empirically evaluate MCPP on CodeFlow and ProofFlow, demonstrating consistent gains over strong baselines across diverse budget–deadline constraints, and further validate its robustness under noisy estimates of subtask success rates and generation lengths.

2 Related Work

Resource-Aware Agent Execution. As LLM systems move from single-turn responses to long-horizon agentic execution, a central question is how to obtain strong task performance without invoking expensive computation at every step. Early work studies this problem at the query level: routing and cascading methods select among models, or call them sequentially, to balance response quality and inference cost [Chen et al., Ding et al., Ong et al., Wang et al., 2025d, Shen et al., 2025, Jali et al., 2026]. This query-level view has recently been extended to more structured and agentic settings, where resource decisions must be made repeatedly during execution. AgentTTS allocates test-time compute across stages of complex tasks [Wang et al., 2025a], and EvoRoute dynamically selects LLM backbones for agentic subtasks to improve the performance–cost–latency frontier [Zhang et al., 2026b]. Another thread makes latency and wall-clock time explicit, showing that token-efficient inference and time-efficient execution can lead to different strategies [Huang et al., 2025, Wang et al., 2025h, Ma et al., 2026, Fan et al., 2026]. More recent agentic methods further introduce explicit budget control, including budget-aware tool scaling, strict-budget tool-use planning, and budget-aware model routing [Liu et al., 2025, 2026, Zhang et al., 2026a, Wu et al., 2026].

This line of work makes agent execution increasingly resource-aware, but it typically optimizes decisions along a single sequential run, such as choosing the next model, tool, compute budget, or

inference behavior based on the current history. Our work studies a different decision unit: Given a dependency-structured workflow, multiple subtasks may become executable simultaneously, and the executor must allocate base models and parallel samples across them while jointly managing the remaining budget and time. We therefore optimize the probability that the entire workflow completes within the specified constraints, rather than a per-step routing decision or an average efficiency frontier.

Performance and Length Estimation. Our framework assumes access to estimates of single-attempt success probability and expected generation length for each subtask–model pair. These quantities are closely related to two lines of work: performance estimation for deciding which model or strategy to use, and length estimation for predicting the cost and latency of an execution. LLM routing methods commonly estimate model quality, query difficulty, confidence, or strategy utility to decide which model or reasoning procedure should be selected [Ding et al., Wang et al., 2025d, Pan et al., 2025, Ding et al., 2025, Shen et al., 2025]. Some methods further adapt the amount of test-time computation, such as choosing the number of sampled responses or selecting among decoding strategies based on predicted performance, token usage, cost, or latency [Ding et al., 2025, Huang et al., 2025, Xie et al., 2026]. Complementary work predicts model success from historical evaluations, small reference sets, or online feedback, reducing the cost of estimating how likely a model is to solve unseen instances [Pacchiardi et al., 2024, Panda et al., 2025]. These studies provide natural sources for the success-rate and generation-length estimates used in our setting. Rather than improving these estimators themselves, we study how to use their outputs for online resource allocation over a dependency-structured workflow, and we evaluate robustness when the estimates are noisy.

Workflow Orchestration. A separate line of work studies the upstream question of how agentic workflows should be designed, generated, or optimized. Graph-based formulations represent agents, tools, and model calls as nodes connected by information-flow or dependency edges, turning agent systems into optimizable computational graphs [Zhuge et al., 2024]. Automated workflow search methods such as AFlow optimize executable workflow programs using task feedback [Zhang et al., 2025c], while subsequent work explores multi-agent architecture search, communication topology design, dynamic agent elimination, workflow evolution, and preference-based workflow optimization [Zhang et al., 2025a,b, Wang et al., 2025g,e,f]. System-oriented efforts further emphasize hierarchical orchestration, workflow serving, distributed execution, and runtime control for long-horizon agent applications [Zhang et al., 2025d, Team et al., 2026, Laju et al., 2026, Zhang et al., 2026c]. Together, this literature establishes workflow structure as a first-class object in agentic systems. Our work is orthogonal to workflow construction: we begin after a workflow has been specified, and study how to execute it under explicit budget and deadline constraints.

3 Constraint-Driven Online Resource Allocation

We now formalize the execution-time allocation problem studied in this work. The workflow is assumed to be fixed before execution begins, which is produced by a human designer, an agentic orchestrator, or an automated workflow search method. Our focus is the following online decision problem:

> *Given a workflow, a budget B, a deadline D, and success-rate and generation-length estimates for each subtask–model pair, how should the executor allocate models and parallel samples to currently executable subtasks so as to maximize the probability that the entire workflow completes within B and D?*

3.1 Problem Formulation

Workflow, models, and estimates. A workflow execution instance is defined as

$$\mathcal{I} = (G, \mathcal{M}, \Phi, B, D),$$

where $G = (V, E)$ is a directed acyclic graph of subtasks, $\mathcal{M}$ is the set of available base models, Φ denotes estimated success and length statistics, B is the total budget, and D is the deadline. Each node $v \in V$ is a subtask, and an edge $(u, v) \in E$ means that v can only be attempted after u has been completed. For a completed set $S \subseteq V$, the currently executable subtasks are

$$R(S) = \{v \in V \setminus S : \mathrm{Pred}(v) \subseteq S\},$$

where $\mathrm{Pred}(v)$ denotes the predecessors of v. The workflow is complete when $S = V$.

For each subtask–model pair (v, m), we assume estimated or calibrated statistics

$$\phi_{v,m} = (p_{v,m}, \ell_{v,m}),$$

where $p_{v,m} \in [0, 1]$ is the single-attempt completion probability and $\ell_{v,m}$ is the expected generation length. We write these estimated quantities without hats for notational simplicity. The generation length is converted into per-attempt cost and latency using model-specific pricing and throughput, denoted by $c_{v,m}$ and $\tau_{v,m}$, respectively. If k independent samples are launched in parallel for subtask v using model m, the probability that at least one sample succeeds is

$$q_{v,m}(k) = 1 - (1 - p_{v,m})^k.$$

Because the marginal gain of additional samples decreases geometrically, we restrict sampling widths to a finite log-scale set $\mathcal{K}$ of positive values (Section C.2 gives a concrete choice of $\mathcal{K}$).

Online state, action, and transition. At any execution step, the state is

$$s = (S, b, h),$$

where $S \subseteq V$ is the set of completed subtasks, b is the remaining budget, and h is the remaining time before the deadline. The initial state is $s_0 = (\emptyset, B, D)$. At state $s = (S, b, h)$, an allocation action assigns a model and a sampling width to every currently executable subtask:

$$a = \{(m_v, k_v)\}_{v \in R(S)}, \qquad m_v \in \mathcal{M}, \quad k_v \in \mathcal{K}.$$

Thus, the log-scale action space is

$$\mathcal{A}_{\mathcal{K}}(S) = \left\{ \{(m_v, k_v)\}_{v \in R(S)} : m_v \in \mathcal{M},\ k_v \in \mathcal{K} \right\}.$$

The budget consumed by action a is

$$C(a) = \sum_{v \in R(S)} k_v c_{v,m_v}.$$

Since currently executable subtasks are executed concurrently, the wall-clock duration is

$$\Delta(a) = \max_{v \in R(S)} \Delta_{v,m_v}(k_v),$$

where $\Delta_{v,m_v}(k_v)$ is the estimated duration of launching k_v parallel samples for subtask v with model m_v. Under ideal parallelism, $\Delta_{v,m_v}(k_v)$ is close to the single-attempt latency τ_{v,m_v}, while more general systems may use an estimated batch-latency model.

If $C(a) > b$ or $\Delta(a) > h$, the action violates the remaining constraints and has constrained completion value zero. Otherwise, after executing a, each currently executable subtask $v \in R(S)$ completes independently with probability $q_{v,m_v}(k_v)$. Let $U \subseteq R(S)$ be the subset of subtasks completed in this round. Then

$$\Pr(U \mid s, a) = \prod_{v \in U} q_{v,m_v}(k_v) \prod_{v \in R(S) \setminus U} \left(1 - q_{v,m_v}(k_v)\right),$$

and the next state is

$$s' = \big(S \cup U,\ b - C(a),\ h - \Delta(a)\big).$$

This transition captures the core difficulty of workflow execution: one allocation can make partial progress, consume budget and time, and unlock different downstream subtasks depending on stochastic outcomes.

Objective and exact Bellman backup. A policy π maps each state $s = (S, b, h)$ to an allocation action. Let T_{solve} be the wall-clock time at which all subtasks are completed, and let C_π be the total execution cost under policy π. We aim to maximize the constrained completion probability

$$J(\pi) = \Pr_{\pi}(T_{\mathrm{solve}} \le D,\ C_\pi \le B).$$

Budget and time are therefore part of the success condition: an execution that exceeds either limit is not merely inefficient, but fails to satisfy the user's request.

Within the log-scale action space $\mathcal{A}_{\mathcal{K}}$, the optimal constrained completion probability admits a dynamic-programming characterization. Let $V^\star_{\mathcal{K}}(S, b, h)$ denote the optimal value from state (S, b, h). The terminal conditions are

$$V^\star_{\mathcal{K}}(S, b, h) = 1 \quad \text{if } S = V,$$

and

$$V^\star_{\mathcal{K}}(S, b, h) = 0 \quad \text{if } S \neq V \text{ and there is no } a \in \mathcal{A}_{\mathcal{K}}(S) \text{ such that } C(a) \leq b,\ \Delta(a) \leq h.$$

For all other states,

$$V^\star_{\mathcal{K}}(S, b, h) = \max_{a \in \mathcal{A}_{\mathcal{K}}(S)} Q^\star_{\mathcal{K}}(s, a),$$

where

$$Q^\star_{\mathcal{K}}(s, a) = \begin{cases} \displaystyle\sum_{U \subseteq R(S)} \Pr(U \mid s, a) V^\star_{\mathcal{K}}\big(S \cup U,\ b - C(a),\ h - \Delta(a)\big), & C(a) \leq b,\ \Delta(a) \leq h, \\ 0, & \text{otherwise.} \end{cases}$$

This Bellman backup makes the allocation trade-off explicit. Using stronger models or more samples can increase immediate progress, but it also consumes budget and time that may be needed by downstream subtasks. Exact dynamic programming is conceptually clean but impractical: the completed set S can take exponentially many values in $|V|$, the action space scales as

$$(|\mathcal{M}|\,|\mathcal{K}|)^{|R(S)|},$$

and each action induces a distribution over $2^{|R(S)|}$ possible success subsets. This combinatorial barrier motivates an online approximation that preserves the constrained completion objective without exhaustive Bellman evaluation.

3.2 Monte Carlo Portfolio Planning

To obtain a practical solver, we propose *Monte Carlo Portfolio Planning* (MCPP), a lightweight online planner that approximates the Bellman decision rule with a one-step portfolio rollout backup (Algorithm 1). At state $s = (S, b, h)$, MCPP constructs a compact candidate action set

$$\widetilde{\mathcal{A}}(s) \subseteq \mathcal{A}_{\mathcal{K}}(S).$$

This set may enumerate the full log-scale action space when few subtasks are executable, or be pruned to a manageable subset when the branching factor is large. In all cases, it includes the base actions induced by a portfolio of simple continuation policies:

$$\Pi_0 = \{\pi_{m,k} : m \in \mathcal{M},\ k \in \mathcal{K}\}.$$

Each $\pi_{m,k}$ is a model-specific Retry-k policy that assigns model m and k parallel samples to every currently executable subtask at each future state:

$$\pi_{m,k}(S, b, h) = \{(m, k)\}_{v \in R(S)}.$$

This portfolio spans conservative to aggressive spending behaviors and repeats this coverage for each available base model.

For a candidate current action $a \in \widetilde{\mathcal{A}}(s)$ and continuation policy $\mu \in \Pi_0$, define

$$Q_\mu(s, a) = \Pr(\text{workflow completes within } b, h \mid \text{first execute } a,\ \text{then follow } \mu)\,.$$

MCPP estimates this probability by Monte Carlo simulation. Each simulated rollout first applies a, samples the stochastic completed subset, updates the state, and then follows μ until the workflow completes or the constraints are violated. Let $\widehat{Q}_\mu(s, a)$ be the resulting Monte Carlo estimate. The candidate action is scored by its best continuation policy,

$$\widehat{Q}_{\Pi_0}(s, a) = \max_{\mu \in \Pi_0} \widehat{Q}_\mu(s, a),$$

and MCPP selects

$$a_{\mathrm{MCPP}}(s) = \arg\max_{a \in \widetilde{\mathcal{A}}(s)} \widehat{Q}_{\Pi_0}(s, a).$$

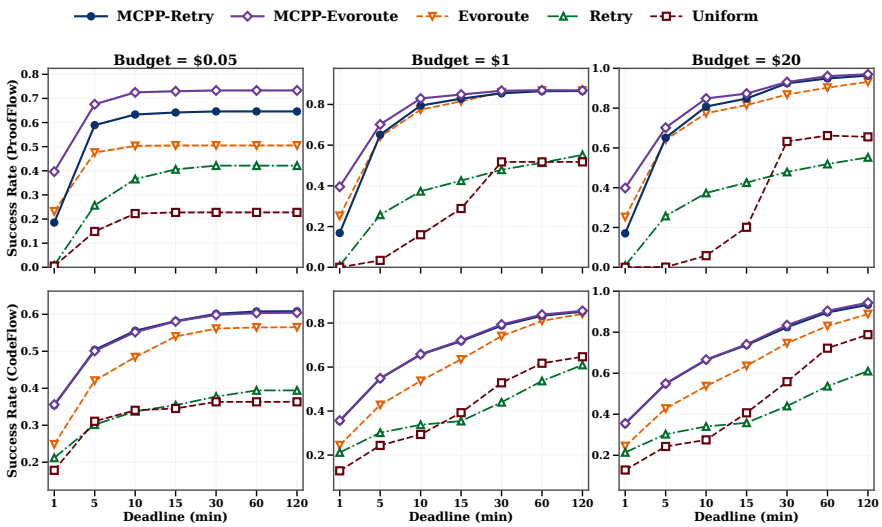

Figure 2: **Main results across workflow benchmarks.** We evaluate Monte Carlo Portfolio Planning (MCPP) under the same budget–deadline settings on ProofFlow and CodeFlow.

Only the selected current action is executed in the real workflow. After observing which subtasks actually completed, the executor updates the state and replans. Thus, MCPP induces a closed-loop execution policy rather than a fixed model assignment or fixed sampling-width rule.

We provide a theoretical analysis showing that, when the candidate set contains the portfolio-induced base actions, MCPP safely improves over the best base policy, up to finite-sample Monte Carlo error, candidate-set approximation error, and errors in the estimated success rates and generation lengths (Appendix B).

4 Experiments

In this section, experimental evaluation results are presented to demonstrate the performance of MCPP for deadline-constrained execution of agentic workflows. We begin by outlining the experiment setup, followed by the main results comparing MCPP against baselines, and analysis focusing on its scalability, efficiency, action diversity, runtime overhead, and robustness to noise disturbance.

4.1 Experimental Setup

We evaluate our method on two dependency-structured workflow benchmarks: ProofFlow [Cabral et al., 2025] and CodeFlow [Wang et al., 2025c]. ProofFlow evaluates formal-reasoning workflows in which an input problem is decomposed into dependent formalization and proving subtasks, while CodeFlow evaluates code-generation workflows in which a programming task is decomposed into dependent implementation and verification subtasks. To accurately evaluate different allocation policies, we first construct empirical execution profiles through large-scale rollouts to capture the statistics of each model–subtask pair, such as success rate and output-token length. Then, numerous offline experiments are conducted to enable statistically reliable evaluation of stochastic workflow execution to measure completion probability under identical budget–deadline settings. More details about the offline workflow pool construction are provided in Appendix C.1, while the action space of MCPP for these two benchmarks is detailed in Appendix C.2.

We compare against three representative baselines. **Uniform** is a static allocation baseline that distributes the available budget uniformly across subtasks before execution and dispatches the assigned rollouts when subtasks become ready. It represents a static strategy that respects workflow dependencies and hard budget constraints, but does not adapt to intermediate outcomes. We report the best Uniform result over available model families. **Retry** is an event-driven online baseline that observes execution events and applies a fixed local retry-width rule to ready subtasks. It represents

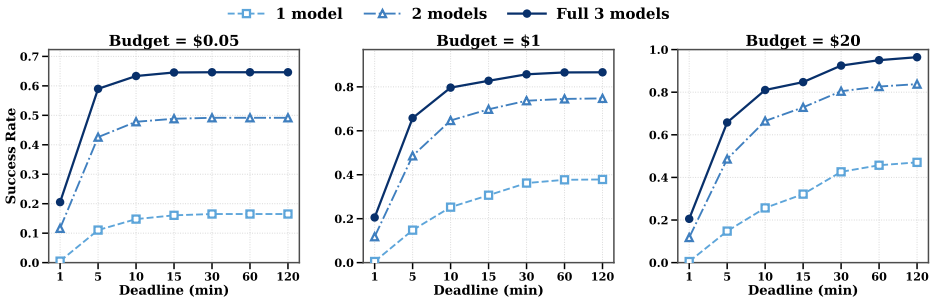

Figure 3: **Effect of model portfolio size.** The full three-model portfolio consistently outperforms smaller portfolios, showing that model diversity improves workflow-level constrained execution.

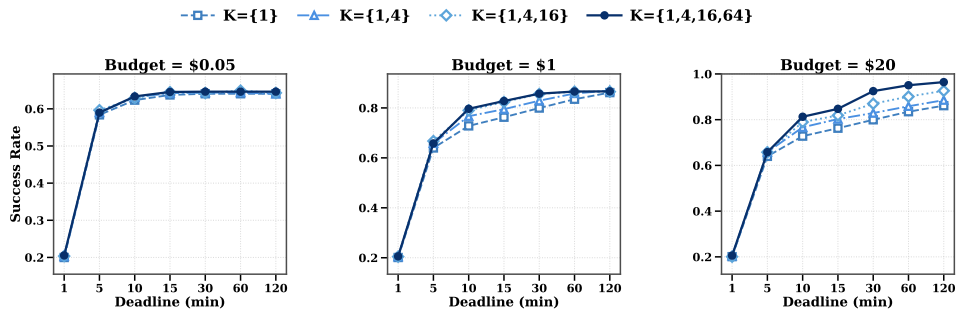

Figure 4: **Effect of rollout-width portfolio.** Richer rollout-width portfolios improve performance by allowing the planner to adapt compute intensity across nodes and states.

the natural alternative of online local retrying without downstream planning. We report the best Retry result over both model families and fixed retry-width choices. **EvoRoute** [Zhang et al., 2026b] is a routing baseline that selects model–rollout-width configurations for individual subtasks according to a performance–cost–latency objective. It represents route-aware resource allocation without MCPP's portfolio lookahead over downstream workflow completion. We report the best EvoRoute result over the evaluated weight configurations.

4.2 Main Results

As shown in Figure 2, MCPP consistently maintains a clear advantage over its corresponding base policy in all budget regimes for both formal-reasoning and code-generation workflows. The improvement is particularly significant under the tightest budget of $B = \$0.05$. With deadline-aware lookahead, the proposed method can rapidly reach a high success rate once the deadline becomes moderately feasible, while the baselines remain substantially lower. Under moderate and loose budgets, i.e., $B = \$1$ and $B = \$20$, our method can also converge rapidly, achieving a success rate close to 1 within the 120-minute deadline. The gains come from planning under joint budget–deadline constraints. To complete a workflow under hard constraints, the executor must decide not only whether a ready subtask should be attempted, but also how much budget and time should be spent now versus preserved for downstream dependencies. Uniform allocation ignores this state-dependent trade-off, while Retry reacts to observed execution events but still uses a fixed local rule. EvoRoute selects model routes for subtasks, but does not replan resource allocation through portfolio lookahead as execution unfolds. In contrast, MCPP evaluates current actions by simulating downstream workflow completions under the remaining budget and deadline, allowing the planner to prioritize bottleneck subtasks and preserve resources for future dependencies.

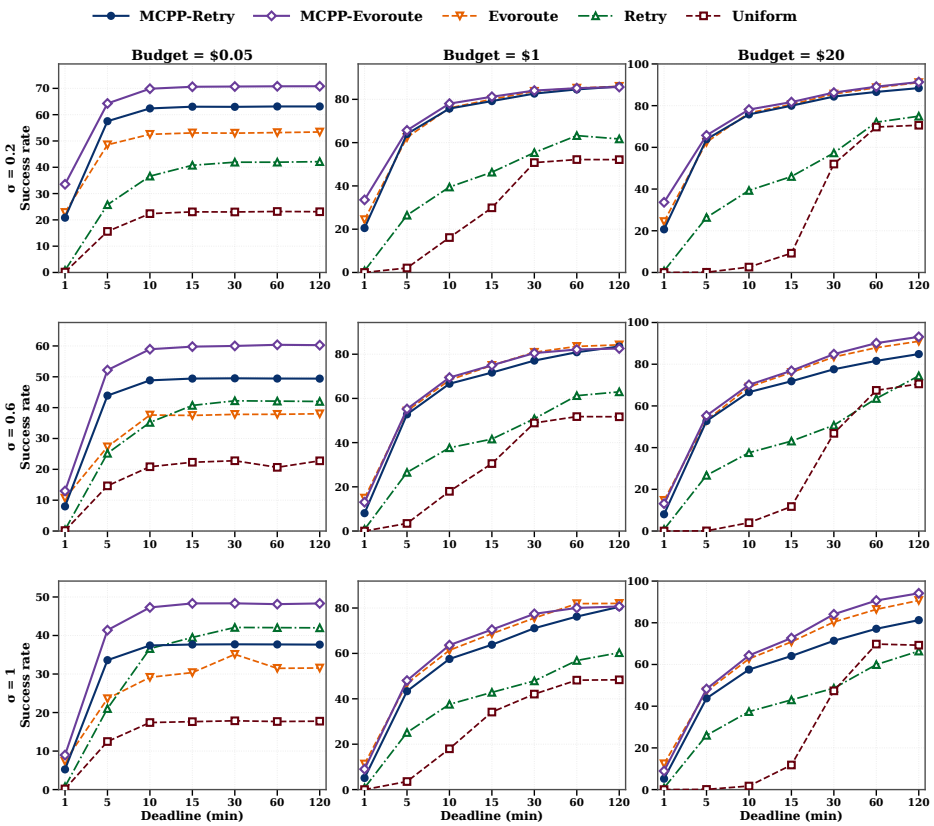

Figure 5: **Robustness to joint success-rate and token-length profile corruption.** MCPP consistently achieves the highest success rate across budgets and corruption levels, demonstrating robustness to inaccurate execution-profile estimates.

4.3 Analysis

We analyze the proposed method from the following perspectives: (1) the effect of action-space diversity, (2) robustness to noisy execution-profile estimates, (3) scaling behavior with workflow size (Appendix D.1), (4) runtime overhead (Appendix D.2), (5) the effect of Monte Carlo searches (Appendix D.3). Unless otherwise specified, analysis experiments are conducted on ProofFlow with Retry as the base policy.

Effect of Action-Space Diversity We first analyze how expanding the planner's action space affects constrained completion probability. The action space contains two complementary dimensions of model choice and rollout-width choice. Specifically, model choice determines the success-latency-cost profile of each attempt, while rollout width determines how aggressively the planner samples the selected model on a ready subtask. Together, these dimensions allow the planner to allocate not only the right model, but also the right amount of computation at each workflow state.

Figure 3 shows that the full three-model portfolio consistently outperforms both the two-model and one-model settings, which indicates that selecting a single globally strongest model is not always sufficient for workflow-level constrained execution. Since model usefulness is subtask-dependent, expensive but strong models can be valuable for difficult bottleneck nodes, while cheaper models can suffice for easier nodes or tighter budgets. Figure 4 further shows that richer rollout-width portfolios improve performance by allowing the planner to vary compute intensity across nodes and states. These results show that the gain comes from action diversity: heterogeneous model families provide different success–latency–cost trade-offs, and multiple rollout widths provide different levels of computational investment. Accordingly, Monte Carlo lookahead then selects among these node–model–rollout actions based on their downstream constrained completion probability.

Robustness to Noisy Estimation To evaluate robustness to imperfect empirical profiles, we simultaneously perturb both success-rate and token-length estimates and compare MCPP, EvoRoute, Retry, and Uniform under exactly the same corrupted-profile realization at every noise level, where actual execution outcomes are still sampled from the true empirical pools. A noise level of 0% uses the original accurate profiles under the same evaluation protocol. At 100% noise, every success-rate estimate is sampled independently from $[0, 1]$, and every token-length estimate is sampled independently from $[0, L_{max}]$, regardless of the corresponding model–subtask pair. The resulting profiles are therefore purely random and completely uninformative. The detailed perturbation approaches are provided in Appendix C.3.

Figure 5 shows that MCPP achieves the highest mean success rate across all evaluated budgets, deadlines, and corruption levels. As expected, its absolute performance decreases as the profiles become less informative, but its advantage over base policies persists throughout the entire corruption range, including the 100% pure-random endpoint. Thus, it can be directly demonstrated that MCPP's relative allocation advantage does not depend on receiving accurate profile information than the competing methods.

5 Conclusions

In this work, we formulate *constraint-driven online resource allocation for agentic workflows*, a hard-constrained execution setting in which the objective is not to optimize an average performance–cost–latency frontier, but to maximize the probability that a given workflow completes successfully within a specified budget and deadline. We model this problem as a finite-horizon stochastic online allocation process over dependency-structured workflows, where the executor dynamically allocates base models and parallel samples to currently executable subtasks according to the remaining budget, remaining time, and observed execution outcomes. To obtain a practical solver, we propose Monte Carlo Portfolio Planning (MCPP), a lightweight closed-loop planner that directly estimates constrained completion probability through simulated workflow executions and replans after each observed outcome. Theoretically, we provide a safe-improvement analysis showing that MCPP improves over the best base policy in its portfolio up to finite-sample, candidate-set, and estimation errors. Empirically, experiments on CodeFlow and ProofFlow demonstrate that MCPP consistently improves constrained completion probability over strong baselines across diverse budget–deadline constraints, with further analyses showing robustness to noisy estimates of subtask success rates and generation lengths.

Limitations and future directions. MCPP relies on estimates of subtask success rates and generation lengths. Although our noise-injection experiments suggest robustness to moderate estimation errors, better calibration and uncertainty-aware estimation could further improve deployment reliability. In addition, our experiments focus on dependency-structured code and proof workflows. Extending this setting to more open-ended tool-use, web, and multi-agent workflows would further clarify the generality of constraint-driven execution under real deployment constraints.

Acknowledgements

This research was supported by the Beijing Natural Science Foundation (No. 4262065), and the National Natural Science Fundation of China (No. 62676029).

References

Aelim Ahn, Sooyeon Lee, Hyosun Wang, Chiwan Park, Daeryong Kim, Jihyeon Roh, Kichang Yang, Wonjun Jang, Hwang Woosung, Min Seok Kim, et al. Orchestrationbench: Llm-driven agentic planning and tool use in multi-domain scenarios. In *The Fourteenth International Conference on Learning Representations*, 2026.

Rafael Cabral, Tuan Manh Do, Xuejun Yu, Wai Ming Tai, Zijin Feng, and Xin Shen. Proofflow: A dependency graph approach to faithful proof autoformalization. *arXiv preprint arXiv:2510.15981*, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks VS Lakshmanan, Qingyun Wu, and Victor Rühle. Best-route: Adaptive llm routing with test-time optimal compute. In *International Conference on Machine Learning*, pages 13870–13884. PMLR, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning*, 2025.

Qi Fan, An Zou, and Yehan Ma. Timebill: Time-budgeted inference for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30620–30628, 2026.

Jenny Y Huang, Mehul Damani, Yousef El-Kurdi, Ramon Astudillo, and Wei Sun. Latency and token-aware test-time compute. *arXiv preprint arXiv:2509.09864*, 2025.

Neharika Jali, Anupam Nayak, and Gauri Joshi. Not all turns are equally hard: Adaptive thinking budgets for efficient multi-turn reasoning. *arXiv preprint arXiv:2604.05164*, 2026.

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.

Marco Laju, Donghyun Son, Saurabh Agarwal, Nitin Kedia, Myungjin Lee, Jayanth Srinivasa, and Aditya Akella. Nalar: An agent serving framework. *arXiv preprint arXiv:2601.05109*, 2026.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025.

Hanbing Liu, Chunhao Tian, Nan An, Ziyuan Wang, Pinyan Lu, Changyuan Yu, and Qi Qi. Budget-constrained agentic large language models: Intention-based planning for costly tool use. *arXiv preprint arXiv:2602.11541*, 2026.

Tengxiao Liu, Zifeng Wang, Jin Miao, I Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations*, 2024.

Yichuan Ma, Linyang Li, Peiji Li, Xiaozhe Li, Qipeng Guo, Dahua Lin, Kai Chen, et al. Timely machine: Awareness of time makes test-time scaling agentic. *arXiv preprint arXiv:2601.16486*, 2026.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2024.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*.

Lorenzo Pacchiardi, Lucy G Cheke, and José Hernández-Orallo. 100 instances is all you need: predicting the success of a new llm on unseen data by testing on a few instances. *arXiv preprint arXiv:2409.03563*, 2024.

Zhihong Pan, Kai Zhang, Yuze Zhao, and Yupeng Han. Route to reason: Adaptive routing for llm and reasoning strategy selection. *arXiv preprint arXiv:2505.19435*, 2025.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive llm routing under budget constraints. *arXiv preprint arXiv:2508.21141*, 2025.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Yuanzhe Shen, Yide Liu, Zisu Huang, Ruicheng Yin, Xiaoqing Zheng, and Xuan-Jing Huang. Sater: A self-aware and token-efficient approach to routing and cascading. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 10526–10540, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Fali Wang, Hui Liu, Jingying Zeng, Zhiwei Zhang, Zongyu Wu, Chen Luo, Zhen Li, Xianfeng Tang, Qi He, Suhang Wang, et al. Agenttts: Large language model agent for test-time compute-optimal scaling strategy in complex tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. 2025b. URL `http://arxiv.org/abs/2504.11354`.

Sizhe Wang, Zhengren Wang, Dongsheng Ma, Yongan Yu, Rui Ling, Zhiyu Li, Feiyu Xiong, and Wentao Zhang. Codeflowbench: A multi-turn, iterative benchmark for complex code generation. *arXiv preprint arXiv:2504.21751*, 2025c.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. Mixllm: Dynamic routing in mixed large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 10912–10922, 2025d.

Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. Evoagentx: An automated framework for evolving agentic workflows. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 643–655, 2025e.

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

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 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24013–24035, 2025g.

Zili Wang, Tianyu Zhang, Haoli Bai, Lu Hou, Xianzhi Yu, Wulong Liu, Shiming Xiang, and Lei Zhu. Faster and better llms via latency-aware test-time scaling. *arXiv preprint arXiv:2505.19634*, 2025h.

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, et al. Contextbudget: Budget-aware context management for long-horizon search agents. *arXiv preprint arXiv:2604.01664*, 2026.

Linqi Xiao, Aparaajitha Gomathinayakam Latha, Venkateswarlu Gudepu, and Andrea Fumagalli. A yang-grounded llm agent supporting multi-vendor openroadm optical transport networks. In *Proceedings of the ACM SIGCOMM 2026 Conference*, pages 2109–2115, 2026.

Huanyi Xie, Yubin Chen, Liangyu Wang, Lijie Hu, and Di Wang. Predicting LLM output length via entropy-guided representations. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Caiqi Zhang, Menglin Xia, Xuchao Zhang, Daniel Madrigal, Ankur Mallick, Samuel Kessler, Victor Ruehle, and Saravan Rajmohan. Budget-aware agentic routing via boundary-guided training. *arXiv preprint arXiv:2602.21227*, 2026a.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *International Conference on Machine Learning*, pages 75834–75852. PMLR, 2025a.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Forty-second International Conference on Machine Learning*, 2025b.

Guibin Zhang, Haiyang Yu, Kaiming Yang, Bingli Wu, Fei Huang, Yongbin Li, and Shuicheng Yan. Evoroute: Experience-driven self-routing llm agent systems. *arXiv preprint arXiv:2601.02695*, 2026b.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025c.

Lei Zhang, Mouxiang Chen, Ruisheng Cao, Jiawei Chen, Fan Zhou, Yiheng Xu, Jiaxi Yang, Zeyao Ma, Liang Chen, Changwei Luo, et al. Megaflow: Large-scale distributed orchestration system for the agentic era. *arXiv preprint arXiv:2601.07526*, 2026c.

Wentao Zhang, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: A hierarchical multi-agent framework for general-purpose task solving. *arXiv e-prints*, pages arXiv–2506, 2025d.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Wendong Xu, Yunzhuo Hao, Songcheng Cai, Xiaochen Wang, et al. Clawbench: Can ai agents complete everyday online tasks? *arXiv preprint arXiv:2604.08523*, 2026d.

Haoran Zheng, ChiaHua Chang, Keqin Li, Jinjin Huang, Ze Yang, and Tianrui Liu. Fingpt-agent: An advanced framework for multimodal research report generation with task-adaptive optimization and hierarchical attention. In *2025 5th International Conference on Neural Networks, Information and Communication Engineering (NNICE)*, pages 1597–1601. IEEE, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

A Monte Carlo Portfolio Planning Algorithm

Algorithm 1 Monte Carlo Portfolio Planning (MCPP)

Require: Workflow instance $\mathcal{I} = (G, \mathcal{M}, \Phi, B, D)$, log-scale sampling widths $\mathcal{K}$, simulation budget N_{sim}.
Ensure: Final execution outcome: SUCCESS or FAILURE.

Initialize $s = (S, b, h) \leftarrow (\emptyset, B, D)$.
Construct portfolio $\Pi_0 \leftarrow \{\pi_{m,k} : m \in \mathcal{M},\ k \in \mathcal{K}\}$.
while $S \neq V$ **do**
    $R(S) \leftarrow \{v \in V \setminus S : \mathrm{Pred}(v) \subseteq S\}$.
    $\widetilde{\mathcal{A}}(s) \leftarrow \textsc{Candidates}(s; \mathcal{M}, \mathcal{K}, \Phi) \cup \{\pi_{m,k}(s) : m \in \mathcal{M},\ k \in \mathcal{K}\}$.
    $\widetilde{\mathcal{A}}_{\mathrm{feas}}(s) \leftarrow \{a \in \widetilde{\mathcal{A}}(s) : C(a) \le b,\ \Delta(a) \le h\}$.
    if $\widetilde{\mathcal{A}}_{\mathrm{feas}}(s) = \emptyset$ **then**
        return FAILURE.
    end if
    for each $a \in \widetilde{\mathcal{A}}_{\mathrm{feas}}(s)$ **do**
        $\widehat{Q}_{\Pi_0}(s, a) \leftarrow \max_{\mu \in \Pi_0} \textsc{MCValue}(s, a, \mu, N_{\mathrm{sim}}; \Phi)$.
    end for
    $a^\star \leftarrow \arg\max_{a \in \widetilde{\mathcal{A}}_{\mathrm{feas}}(s)} \widehat{Q}_{\Pi_0}(s, a)$.
    Execute $a^\star$ in the real workflow and observe completed subtasks $U \subseteq R(S)$.
    Update $S \leftarrow S \cup U, \quad b \leftarrow b - C(a^\star), \quad h \leftarrow h - \Delta(a^\star)$.
end while
return SUCCESS.
function $\textsc{MCValue}(s, a, \mu, N_{\mathrm{sim}}; \Phi)$
    $n_{\mathrm{succ}} \leftarrow 0$.
    for $i = 1$ **to** N_{sim} **do**
        $\tilde{s} \leftarrow \textsc{SimulateOneStep}(s, a; \Phi)$.
        $n_{\mathrm{succ}} \leftarrow n_{\mathrm{succ}} + \textsc{Rollout}(\tilde{s}, \mu; \Phi)$.
    end for
    return $n_{\mathrm{succ}}/N_{\mathrm{sim}}$.
end function

As shown in Algorithm 1, MCPP implements a closed-loop planning procedure: at each state, it constructs candidate actions, estimates their downstream constrained completion probabilities, executes only the selected action, and then replans after observing the actual outcome. The subroutine MCVALUE estimates the value of a candidate action by first simulating that action and then rolling out a continuation policy μ from the portfolio. Here, ROLLOUT returns a binary indicator of whether the simulated continuation completes the workflow within the remaining budget and deadline; if a simulated action violates the remaining constraints, the simulation is counted as a failure. The continuation policy μ is therefore used only for scoring the current candidate action, while the real workflow executes only $a^\star$, observes the completed subtasks, updates the state, and replans.

B Theory: Portfolio Rollout Planning Provides Safe Improvement

We analyze the Monte Carlo portfolio planner used in our framework. The planner can be viewed as a one-step rollout backup: at the current workflow state, it evaluates candidate allocation actions by simulating their future consequences under a portfolio of simple continuation policies. The goal is not to prove global optimality over the full Bellman action space, which is intractable for realistic workflows. Instead, we show that the portfolio planner provides a safe improvement over the best base policy in the portfolio, up to Monte Carlo estimation error, candidate-set approximation error, and errors in the estimated success rates and generation lengths.

B.1 Setup: states, base policies, and rollout values

Consider a workflow execution state

$$s = (S, b, h),$$

where S is the set of completed subtasks, b is the remaining budget, and h is the remaining time before the deadline. Let $R(S)$ denote the set of currently executable subtasks.

Each allocation action assigns a base model and a parallel sampling width to every executable subtask:

$$a = \{(m_v, k_v)\}_{v \in R(S)}, \qquad m_v \in \mathcal{M}, \quad k_v \in \mathcal{K},$$

where $\mathcal{M}$ is the candidate model set and $\mathcal{K}$ is a finite log-scale set of positive sampling widths. The log-scale grid reflects the diminishing marginal return of parallel samples. If a single attempt succeeds with probability p, then k independent samples succeed with probability

$$q(k) = 1 - (1-p)^k,$$

whose marginal gain satisfies

$$q(k+1) - q(k) = p(1-p)^k.$$

Thus, larger sampling widths provide progressively smaller additional gains, making exponentially spaced widths a compact coverage of conservative-to-aggressive allocation regimes. The concrete choice of $\mathcal{K}$ is specified in the experimental setup.

Let $\widetilde{\mathcal{A}}(s)$ be the candidate action set evaluated by the planner at state s. The planner also uses a portfolio of base continuation policies:

$$\Pi_0 = \{\pi_{m,k} : m \in \mathcal{M},\ k \in \mathcal{K}\}.$$

Each $\pi_{m,k}$ is a model-specific Seq-k policy. At every future state $s' = (S', b', h')$, it assigns the same model m and sampling width k to every executable subtask:

$$\pi_{m,k}(s') = \{(m,k)\}_{v \in R(S')}.$$

If this action violates the remaining budget or deadline, its value is defined to be zero.

For any policy $\mu \in \Pi_0$, let $V^\mu(s)$ denote its constrained completion probability from state s:

$$V^\mu(s) = \Pr_\mu\big(\text{the workflow completes within remaining budget } b \text{ and remaining time } h \mid s\big).$$

For a candidate current action $a \in \widetilde{\mathcal{A}}(s)$, define the rollout value under continuation policy μ as

$$Q_\mu(s,a) = \mathbb{E}_{s' \sim P(\cdot \mid s,a)}\left[V^\mu(s')\right],$$

where $P(\cdot \mid s,a)$ is the stochastic transition induced by executing a. The portfolio value of action a is

$$Q_{\Pi_0}(s,a) = \max_{\mu \in \Pi_0} Q_\mu(s,a).$$

B.2 Exact portfolio improvement

The key requirement is that the candidate action set contains the first-step actions of all base policies.

Assumption 1 (Base actions are included)**.** For every $\mu \in \Pi_0$, the action $\mu(s)$ is included in $\widetilde{\mathcal{A}}(s)$. If $\mu(s)$ violates the remaining budget or deadline, it is still treated as a candidate action with value zero.

Definition 2 (Exact portfolio planner)**.** The exact portfolio planner chooses

$$a_{\Pi_0}(s) \in \arg\max_{a \in \widetilde{\mathcal{A}}(s)} Q_{\Pi_0}(s,a) = \arg\max_{a \in \widetilde{\mathcal{A}}(s)} \max_{\mu \in \Pi_0} Q_\mu(s,a).$$

After executing $a_{\Pi_0}(s)$, the planner observes the actual stochastic outcome, updates the workflow state, and replans.

Theorem 3 (State-wise safe improvement over the portfolio)**.** *Under Assumption 1, the exact portfolio planner satisfies*

$$Q_{\Pi_0}\big(s, a_{\Pi_0}(s)\big) \geq \max_{\mu \in \Pi_0} V^\mu(s).$$

Proof. By definition,

$$Q_{\Pi_0}\big(s, a_{\Pi_0}(s)\big) = \max_{a \in \widetilde{\mathcal{A}}(s)} \max_{\nu \in \Pi_0} Q_\nu(s,a).$$

For any $\mu \in \Pi_0$, Assumption 1 implies that $\mu(s) \in \widetilde{\mathcal{A}}(s)$. By the Bellman identity for the fixed policy μ,

$$V^\mu(s) = Q_\mu(s, \mu(s)).$$

Therefore,

$$Q_{\Pi_0}\big(s, a_{\Pi_0}(s)\big) \geq Q_\mu(s,\mu(s)) = V^\mu(s).$$

Taking the maximum over $\mu \in \Pi_0$ proves the claim.

Theorem 3 shows that the exact portfolio backup cannot be worse than simply following the best base policy in Π_0 from the current state. The planner may choose an action that is not itself a Seq-k action, but the value of that action, evaluated under the best continuation policy in the portfolio, is at least the value of the best portfolio policy.

Assumption 4 (Finite execution horizon)**.** The execution process has a finite maximum remaining horizon. Equivalently, from every non-terminal state, the process terminates after at most $H_{\max} < \infty$ additional decision rounds under any feasible policy.

We can also interpret the exact result in the closed-loop setting. Let π_{exact} be the policy that applies the exact portfolio planner at every visited state.

Theorem 5 (Closed-loop portfolio improvement)**.** *Assume Assumptions 1 and 4. Then for any state s,*

$$V^{\pi_{\text{exact}}}(s) \geq \max_{\mu \in \Pi_0} V^{\mu}(s).$$

Proof. We prove the claim by backward induction on the maximum number of remaining execution rounds. The statement is immediate at terminal states.

Consider a non-terminal state s. Let

$$a^\star = a_{\Pi_0}(s)$$

be the exact portfolio action selected at s, and let $\mu^\star \in \Pi_0$ be a continuation policy attaining

$$Q_{\Pi_0}(s, a^\star) = Q_{\mu^\star}(s, a^\star).$$

After executing $a^\star$, the next state s' is drawn from $P(\cdot \mid s, a^\star)$. By the induction hypothesis,

$$V^{\pi_{\text{exact}}}(s') \geq V^{\mu^\star}(s')$$

for every next state s'. Therefore,

$$\begin{aligned} V^{\pi_{\text{exact}}}(s) &= \mathbb{E}_{s' \sim P(\cdot|s,a^\star)} \left[V^{\pi_{\text{exact}}}(s') \right] \\ &\geq \mathbb{E}_{s' \sim P(\cdot|s,a^\star)} \left[V^{\mu^\star}(s') \right] \\ &= Q_{\mu^\star}(s, a^\star) \\ &= Q_{\Pi_0}(s, a^\star) \\ &\geq \max_{\mu \in \Pi_0} V^{\mu}(s), \end{aligned}$$

where the last inequality follows from Theorem 3.

The closed-loop result applies to the exact portfolio planner. For the finite-sample planner, we provide a state-wise guarantee below. A full closed-loop finite-sample guarantee would require uniform concentration over all states visited by the planner.

B.3 Finite-sample Monte Carlo estimation

In practice, the rollout values $Q_\mu(s, a)$ are not available exactly. For each candidate pair (a, μ), the planner runs N independent simulations. Let

$$Y_i(a, \mu) = \mathbf{1}[\text{simulation } i \text{ completes the workflow within the remaining budget and deadline}].$$

Then

$$\widehat{Q}_\mu(s, a) = \frac{1}{N} \sum_{i=1}^{N} Y_i(a, \mu)$$

is an unbiased estimator of $Q_\mu(s, a)$.

The empirical portfolio value of action a is

$$\widehat{Q}_{\Pi_0}(s, a) = \max_{\mu \in \Pi_0} \widehat{Q}_\mu(s, a).$$

The finite-sample planner selects

$$\widehat{a} \in \arg\max_{a \in \widetilde{\mathcal{A}}(s)} \widehat{Q}_{\Pi_0}(s, a).$$

Let

$$\widehat{\mu} \in \arg\max_{\mu \in \Pi_0} \widehat{Q}_{\mu}(s, \widehat{a})$$

be the continuation policy attaining the empirical score for $\widehat{a}$. Only $\widehat{a}$ is executed in the real workflow; $\widehat{\mu}$ is used only to score the current action.

Let

$$L(s) = |\widetilde{\mathcal{A}}(s)|\,|\Pi_0|$$

be the number of action–continuation pairs evaluated at state s.

Theorem 6 (Finite-sample state-wise guarantee)**.** *Fix a state s and confidence level $\delta \in (0, 1)$. With probability at least $1 - \delta$,*

$$Q_{\widehat{\mu}}(s, \widehat{a}) \geq \max_{a \in \widetilde{\mathcal{A}}(s),\, \mu \in \Pi_0} Q_{\mu}(s, a) - 2\epsilon(s, \delta),$$

where

$$\epsilon(s, \delta) = \sqrt{\frac{\log(2L(s)/\delta)}{2N}}.$$

Consequently, under Assumption 1,

$$Q_{\widehat{\mu}}(s, \widehat{a}) \geq \max_{\mu \in \Pi_0} V^{\mu}(s) - 2\epsilon(s, \delta).$$

Proof. For any fixed pair (a, μ), the variables $Y_i(a, \mu)$ are independent Bernoulli random variables with mean $Q_{\mu}(s, a)$. By Hoeffding's inequality,

$$\Pr\left(\left|\widehat{Q}_{\mu}(s, a) - Q_{\mu}(s, a)\right| > \epsilon\right) \leq 2\exp(-2N\epsilon^2).$$

Applying a union bound over all $L(s)$ pairs, with probability at least $1 - \delta$, all estimates satisfy

$$\left|\widehat{Q}_{\mu}(s, a) - Q_{\mu}(s, a)\right| \leq \epsilon(s, \delta).$$

Let

$$(a^{\star}, \mu^{\star}) \in \arg\max_{a \in \widetilde{\mathcal{A}}(s),\, \mu \in \Pi_0} Q_{\mu}(s, a).$$

On the high-probability event,

$$\begin{aligned} Q_{\widehat{\mu}}(s, \widehat{a}) &\geq \widehat{Q}_{\widehat{\mu}}(s, \widehat{a}) - \epsilon(s, \delta) \\ &\geq \widehat{Q}_{\mu^{\star}}(s, a^{\star}) - \epsilon(s, \delta) \\ &\geq Q_{\mu^{\star}}(s, a^{\star}) - 2\epsilon(s, \delta). \end{aligned}$$

This proves the first statement. The second statement follows from Theorem 3.

Theorem 6 gives the finite-sample counterpart of the exact safe-improvement result. The selected action is within a standard Monte Carlo estimation error of the best evaluated action–continuation pair, and therefore within the same error of the best base policy in the portfolio.

B.4 Approximation to the exact Bellman backup

The finite-sample theorem is stated for the candidate action set $\widetilde{\mathcal{A}}(s)$ and the portfolio continuation class Π_0. We now make explicit how these approximations relate to the exact Bellman backup over the full log-scale action space.

Let $\mathcal{A}_{\mathcal{K}}(s)$ denote the full log-scale action space in which each executable subtask chooses a model $m \in \mathcal{M}$ and a sampling width $k \in \mathcal{K}$. The candidate action set satisfies

$$\widetilde{\mathcal{A}}(s) \subseteq \mathcal{A}_{\mathcal{K}}(s).$$

Define the candidate-set gap

$$\eta(s) = \max_{a \in \mathcal{A}_{\mathcal{K}}(s),\, \mu \in \Pi_0} Q_{\mu}(s, a) - \max_{a \in \widetilde{\mathcal{A}}(s),\, \mu \in \Pi_0} Q_{\mu}(s, a).$$

By definition, $\eta(s) \geq 0$, and $\eta(s) = 0$ whenever $\widetilde{\mathcal{A}}(s) = \mathcal{A}_{\mathcal{K}}(s)$.

Corollary 7 (Candidate-set gap)**.** *Under the conditions of Theorem 6, with probability at least* $1-\delta$*,*

$$Q_{\widehat{\mu}}(s,\widehat{a}) \geq \max_{a\in\mathcal{A}_{\mathcal{K}}(s),\,\mu\in\Pi_0} Q_\mu(s,a) - \eta(s) - 2\epsilon(s,\delta).$$

Proof. The result follows by substituting the definition of $\eta(s)$ into Theorem 6.

The gap $\eta(s)$ measures the loss from not evaluating the full log-scale action space. This is still a gap within the portfolio-rollout objective, because the continuation value is restricted to Π_0. We next isolate the additional gap between portfolio continuation and the exact Bellman continuation.

Let $V^\star_{\mathcal{K}}(s)$ denote the optimal constrained completion probability over the full log-scale action space $\mathcal{A}_{\mathcal{K}}$. For any first action $a \in \mathcal{A}_{\mathcal{K}}(s)$, define the exact Bellman action value

$$Q^\star_{\mathcal{K}}(s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[V^\star_{\mathcal{K}}(s')\right].$$

The best log-scale Bellman action value is

$$Q^\star_{\mathcal{K}}(s) = \max_{a\in\mathcal{A}_{\mathcal{K}}(s)} Q^\star_{\mathcal{K}}(s,a).$$

Because $V^\star_{\mathcal{K}}(s') \geq V^\mu(s')$ for every $\mu \in \Pi_0$, we have

$$Q^\star_{\mathcal{K}}(s,a) \geq Q_\mu(s,a) \qquad \forall a,\mu.$$

Define the portfolio continuation gap

$$\zeta(s) = Q^\star_{\mathcal{K}}(s) - \max_{a\in\mathcal{A}_{\mathcal{K}}(s),\,\mu\in\Pi_0} Q_\mu(s,a).$$

This term measures the loss from using the base-policy portfolio as the continuation value class instead of the exact optimal continuation value. It is zero when the portfolio contains an optimal continuation policy for the best log-scale action.

Corollary 8 (Approximation to the log-scale Bellman backup)**.** *Under the conditions of Theorem 6, with probability at least* $1-\delta$*,*

$$Q^\star_{\mathcal{K}}(s,\widehat{a}) \geq Q^\star_{\mathcal{K}}(s) - \zeta(s) - \eta(s) - 2\epsilon(s,\delta).$$

Proof. Since $V^\star_{\mathcal{K}}(s') \geq V^{\widehat{\mu}}(s')$ for every next state s',

$$Q^\star_{\mathcal{K}}(s,\widehat{a}) \geq Q_{\widehat{\mu}}(s,\widehat{a}).$$

By Corollary 7,

$$Q_{\widehat{\mu}}(s,\widehat{a}) \geq \max_{a\in\mathcal{A}_{\mathcal{K}}(s),\,\mu\in\Pi_0} Q_\mu(s,a) - \eta(s) - 2\epsilon(s,\delta).$$

By the definition of $\zeta(s)$,

$$\max_{a\in\mathcal{A}_{\mathcal{K}}(s),\,\mu\in\Pi_0} Q_\mu(s,a) = Q^\star_{\mathcal{K}}(s) - \zeta(s).$$

Combining the inequalities proves the claim.

Corollary 8 decomposes the one-step gap to the exact log-scale Bellman backup into three terms: the portfolio continuation gap $\zeta(s)$, the candidate-set gap $\eta(s)$, and the Monte Carlo estimation error $2\epsilon(s,\delta)$. This decomposition clarifies that the planner is not claimed to be globally optimal. Its guarantee is strongest when the base portfolio provides good continuation policies, the candidate actions cover useful allocations, and the Monte Carlo budget is sufficiently large.

B.5 Effect of imperfect success and length estimates

The analysis so far assumes that rollout simulations use the true success, cost, and latency statistics. In practice, subtask success rates and generation lengths are estimated from historical executions, offline calibration, or learned predictors. We now state a robustness bound that separates Monte Carlo estimation error from estimate-induced value error.

Let P denote the true transition kernel and let $\widetilde{P}$ denote the transition kernel induced by the estimated success and length statistics used in simulation. For any candidate pair (a,μ), let $Q^P_\mu(s,a)$

and $Q_\mu^{\widetilde{P}}(s,a)$ denote the corresponding rollout values. Assume that the estimate-induced value discrepancy is uniformly bounded:

$$\left|Q_\mu^{\widetilde{P}}(s,a) - Q_\mu^P(s,a)\right| \le \beta(s), \qquad \forall a \in \widetilde{\mathcal{A}}(s),\ \mu \in \Pi_0.$$

The quantity $\beta(s)$ captures the value error caused by imperfect estimates of success rates, generation lengths, costs, or latencies.

Theorem 9 (Finite-sample guarantee under estimation error)**.** *Suppose Monte Carlo simulations are generated using the estimated transition kernel $\widetilde{P}$, and the estimate-induced value discrepancy is bounded by $\beta(s)$. With probability at least $1-\delta$,*

$$Q_{\widehat{\mu}}^P(s,\widehat{a}) \ge \max_{a \in \widetilde{\mathcal{A}}(s),\ \mu \in \Pi_0} Q_\mu^P(s,a) - 2\epsilon(s,\delta) - 2\beta(s).$$

Consequently, under Assumption 1,

$$Q_{\widehat{\mu}}^P(s,\widehat{a}) \ge \max_{\mu \in \Pi_0} V_P^\mu(s) - 2\epsilon(s,\delta) - 2\beta(s).$$

Proof. The Monte Carlo estimates concentrate around $Q_\mu^{\widetilde{P}}(s,a)$. By Theorem 6, applied under the simulated transition kernel $\widetilde{P}$, with probability at least $1-\delta$,

$$Q_{\widehat{\mu}}^{\widetilde{P}}(s,\widehat{a}) \ge \max_{a,\mu} Q_\mu^{\widetilde{P}}(s,a) - 2\epsilon(s,\delta).$$

Using the estimate-error bound,

$$\begin{aligned} Q_{\widehat{\mu}}^P(s,\widehat{a}) &\ge Q_{\widehat{\mu}}^{\widetilde{P}}(s,\widehat{a}) - \beta(s) \\ &\ge \max_{a,\mu} Q_\mu^{\widetilde{P}}(s,a) - 2\epsilon(s,\delta) - \beta(s) \\ &\ge \max_{a,\mu} Q_\mu^P(s,a) - 2\epsilon(s,\delta) - 2\beta(s). \end{aligned}$$

The portfolio comparison follows from Theorem 3.

Theorem 9 explains the role of the noise-injection experiments. When success rates or generation lengths are estimated imperfectly, the planner optimizes the simulated workflow model induced by these estimates. The degradation is controlled by the induced value discrepancy $\beta(s)$. Empirically, we perturb predicted subtask success probabilities and generation lengths to test how sensitive the planner is to such estimation errors.

A sufficient condition for success-probability errors. A concrete sufficient condition for small $\beta(s)$ can be stated for errors in subtask completion probabilities, assuming the cost and latency estimates are fixed. Errors in generation length, cost, or latency can also affect feasibility boundaries. These effects are captured by the abstract value discrepancy $\beta(s)$ above, or can be analyzed under additional discretization or regularity assumptions on the value function.

Suppose only the subtask completion probabilities are perturbed. If, over the remaining execution horizon $H(s)$,

$$\sup_{s,a} \mathrm{TV}\left(P(\cdot \mid s,a), \widetilde{P}(\cdot \mid s,a)\right) \le \alpha,$$

where TV denotes total variation distance, then a standard simulation argument gives

$$\left|V_P^\pi(s) - V_{\widetilde{P}}^\pi(s)\right| \le \min\{1, H(s)\alpha\}$$

for any fixed policy π. Thus, in Theorem 9, one may take

$$\beta(s) \le \min\{1, H(s)\alpha\}.$$

For the Bernoulli completion model, suppose each attempted subtask success probability is perturbed by at most ρ after accounting for the selected model and sampling width:

$$|q_{v,m}(k) - \widetilde{q}_{v,m}(k)| \le \rho.$$

For an action attempting r executable subtasks, a coupling argument yields

$$\mathrm{TV}\left(P(\cdot \mid s,a), \widetilde{P}(\cdot \mid s,a)\right) \le \min\{1, r\rho\}.$$

This bound is conservative, but it shows explicitly how errors in predicted subtask success probabilities affect rollout evaluation. Errors in generation-length estimates are handled by the more general value-discrepancy term $\beta(s)$.

B.6 Estimating the closed-loop completion probability

The same Monte Carlo mechanism can be used to estimate the success probability of the full closed-loop planner. Given a workflow, a budget B, and a deadline D, run N_{eval} simulated executions of the complete online policy from

$$s_0 = (\emptyset, B, D).$$

Let

$$Z_i = \mathbf{1}[\text{closed-loop simulation } i \text{ completes within } B, D].$$

The predicted constrained completion probability is

$$\widehat{P}_{\mathrm{succ}} = \frac{1}{N_{\mathrm{eval}}} \sum_{i=1}^{N_{\mathrm{eval}}} Z_i.$$

Since Z_i are Bernoulli variables, Hoeffding's inequality gives

$$\Pr\left(\left|\widehat{P}_{\mathrm{succ}} - P_{\mathrm{succ}}\right| > \epsilon\right) \le 2\exp(-2N_{\mathrm{eval}}\epsilon^2).$$

Equivalently, with probability at least $1-\delta$,

$$\left|\widehat{P}_{\mathrm{succ}} - P_{\mathrm{succ}}\right| \le \sqrt{\frac{\log(2/\delta)}{2N_{\mathrm{eval}}}}.$$

This prediction mode estimates the probability that the full online planner completes the workflow within the specified budget and deadline. When simulations use estimated success rates and generation lengths, this is an estimate under the induced simulated execution model. Calibration against real executions is evaluated empirically. It is distinct from the action-selection guarantees above, which compare a single planning decision or the exact closed-loop planner against the base-policy portfolio.

C Implementation Details

C.1 Offline Workflow Pool Construction

We construct empirical execution profiles through large-scale offline rollouts. For each model–subtask pair, the rollout pool stores 512 samples with success outcomes, wall-clock latency, and output-token length. For each benchmark, we build the evaluation pool from the union of usable workflows obtained from the evaluated model rollouts, resulting in 103 usable workflows for ProofFlow and 462 usable workflows for CodeFlow. All experiments for collecting offline model rollouts are executed in parallel on a cluster with 512 NVIDIA H800 GPUs, where each individual run is allocated to 8 GPUs.

For ProofFlow, offline rollout collection requires both formalization and proof generation. On each 8-H800 machine, we split the GPUs into two groups of four GPUs: one group loads the formalizer model and the other group loads the prover model. A workflow rollout first invokes the formalizer to translate the input problem into a Lean statement, and then invokes the prover to generate the corresponding proof. Lean 4.15.0 is utilized as the verifier to check whether the generated proof is accepted.

To ensure reproducibility and fair comparison, we adhered to the sampling hyperparameters for each model family:

- Qwen3-Thinking-Series: Temperature $T = 0.6$, top-$p = 0.95$, top-$k = 20$, with a maximum generation length of 38k tokens.

- Qwen3-Instruct-Series: Temperature $T = 0.7$, top-$p = 0.8$, top-$k = 20$, with a maximum generation length of 38k tokens.
- Goedel-V2-Series: Temperature $T = 0.6$, top-$p = 0.95$, top-$k = 20$, with a maximum generation length of 16k tokens.
- Kimina-7B-Series: Temperature $T = 0.6$, top-$p = 0.95$, top-$k = 0$, with a maximum generation length of 8k tokens.

C.2 Online Simulation

All simulator evaluations of the proposed MCPP method are CPU-only and are conducted on machines equipped with AMD EPYC 9T24 96-Core processors with 96 CPU cores. The inner Monte Carlo rollout evaluator is implemented with a Numba-accelerated kernel to reduce Python overhead. For ProofFlow, we choose three model pairs as our candidate model set: GOEDEL-FORMALIZER-V2-32B with GOEDEL-PROVER-V2-32B, GOEDEL-FORMALIZER-V2-8B with GOEDEL-PROVER-V2-8B [Lin et al., 2025], and KIMINA-AUTOFORMALIZER-7B with KIMINA-PROVER-PREVIEW-DISTILL-7B [Wang et al., 2025b]. For CodeFlow, we choose four models as our candidate model set: QWEN3-30B-A3B-INSTRUCT-2507, QWEN3-30B-A3B-THINKING-2507, QWEN3-4B-INSTRUCT-2507, and QWEN3-4B-THINKING-2507 [Yang et al., 2025]. Moreover, we employ $\mathcal{K} = \{1, 4, 16, 64\}$ as our candidate rollout-width portfolio. We calculate budget consumption according to official output-token prices[3].

C.3 Noise Perturbation Details

In the robustness experiments, we perturb only the planner-visible empirical profiles while keeping the true simulator execution distribution unchanged. For each noise level $\sigma \in \{0.2, 0.6, 1.0\}$, the planner uses a fixed noisy profile, whereas execution outcomes are sampled from the original empirical profiles. Thus, the experiment measures robustness to a mismatch between the statistics used for planning and the distribution encountered during execution.

For token length, let $T_i^{(v,m)}$ denote the token length of the i-th empirical rollout for node v and model m. Let

$$T_{\min} = \min_{v,m,i} T_i^{(v,m)}, \qquad T_{\max} = \max_{v,m,i} T_i^{(v,m)}$$

be the global minimum and maximum over the complete profile pool. We perturb each planner-visible token length according to

$$\widetilde{T}_i^{(v,m)} = \mathrm{clip}\left(T_i^{(v,m)} + \sigma T_{\max} z_i^{(v,m)},\, T_{\min},\, T_{\max}\right), \qquad z_i^{(v,m)} \sim \mathcal{N}(0,1).$$

The same per-sample ratio $r_i^{(v,m)} = \widetilde{T}_i^{(v,m)} / T_i^{(v,m)}$ is then used to scale the corresponding planner-visible latency and cost statistics, so that token length, latency, and cost remain coupled.

For success probability, let $p_{v,m}$ be the empirical success rate of node v under model m. We perturb it in probability space:

$$\widetilde{p}_{v,m} = \mathrm{clip}\left(p_{v,m} + \sigma z_{v,m},\, \epsilon,\, 1-\epsilon\right), \qquad z_{v,m} \sim \mathcal{N}(0,1),$$

where ϵ is a small numerical constant. Given $n_{v,m}$ empirical rollouts, success labels are minimally flipped so that the number of successful samples is

$$\widetilde{n}_{v,m}^{+} = \mathrm{round}\left(\widetilde{p}_{v,m} n_{v,m}\right).$$

Consequently, the planner observes a noisy success and resource profile, while the simulator continues to execute using the unmodified empirical outcomes.

For every nonzero σ, we persist five independently generated noisy profiles and evaluate each profile with 100 independent execution seeds. The $\sigma = 0$ control uses one unperturbed profile with 100 execution seeds. All methods use the corresponding noisy profile for planning or candidate selection and the true profile for execution.

[3] https://www.alibabacloud.com/help/en/model-studio/model-pricing

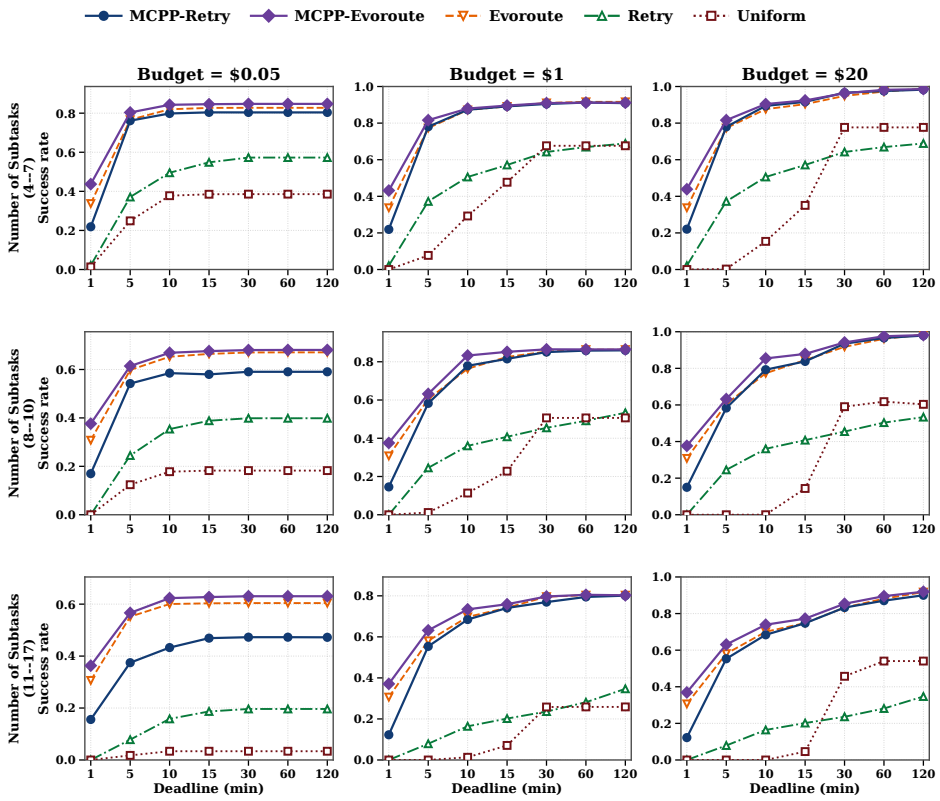

Figure 6: **Scaling with the size of workflows.** Monte Carlo Portfolio Planning (MCPP) generally outperforms baselines across workflow sizes, with particularly clear gains on larger and tighter instances.

D Additional Analysis

D.1 Scaling with Workflow Size

To assess whether our method remains effective on larger workflows, we group ProofFlow workflows by the number of subtasks and report the success rates for different sizes in Figure 6. Figure 6 shows that MCPP-EvoRoute is generally the strongest method across all workflow sizes, while the MCPP variants consistently maintain a clear advantage over their corresponding base policies. These advantages are most pronounced for larger workflows under tight budget–deadline constraints. As workflow size increases, Retry and Uniform degrade sharply, since they cannot fully adapt to realized intermediate outcomes. In contrast, by updating its allocation state after each execution event, MCPP remains more robust to the accumulated effects of early allocation errors.

D.2 Runtime Analysis

To quantify the overhead of online planning, we report the mean planner wall-clock time under the default $M = 64$ configuration on CodeFlow in Table 1. Planning overhead remains moderate across all budget–deadline settings. Under tight constraints, the mean planner time is around 0.6 seconds, since only a small set of actions is feasible and simulated continuations terminate quickly. As the budget and deadline are relaxed, more actions become feasible and Monte Carlo continuations can explore deeper into the remaining workflow, leading to a predictable increase in runtime. However, even in the loosest setting, the mean planner time remains only a few seconds, indicating that the default implementation is practical for online receding-horizon execution.

Table 1: Planner wall-clock time across different budget–deadline settings on CodeFlow.

Budget (B)	**Deadline** (D)						
	1 min	5 min	10 min	15 min	30 min	60 min	120 min
\$0.05	0.64	0.82	0.91	0.97	1.06	1.07	1.11
\$1	0.60	0.88	1.07	1.30	1.79	2.38	3.09
\$20	0.60	0.89	1.09	1.36	1.96	2.85	4.30

Table 2: Constrained completion probability and cumulative MCPP planning time under different Monte Carlo budgets on ProofFlow. Each cell is formatted as mean completion probability (mean cumulative planning time per workflow execution), reported in percentage and seconds, respectively. For each execution, cumulative planning time is obtained by summing the wall-clock durations of all MCPP planning calls made during that execution and is then averaged across executions.

Budget	**MC samples**	$D = 1$ min	$D = 5$ min	$D = 10$ min	$D = 15$ min	$D = 30$ min	$D = 60$ min	$D = 120$ min
$B = \$0.05$	$M = 16$	19.9 (0.92)	55.5 (1.03)	61.9 (1.11)	63.2 (1.16)	63.4 (1.09)	63.4 (1.16)	63.4 (1.07)
	$M = 32$	20.2 (1.44)	57.4 (1.75)	62.4 (1.69)	64.0 (1.60)	64.1 (1.75)	64.1 (1.77)	64.1 (1.77)
	$M = 64$	20.5 (2.40)	59.0 (2.84)	63.3 (2.94)	64.5 (2.88)	64.6 (3.01)	64.6 (2.82)	64.6 (3.02)
	$M = 128$	20.3 (4.30)	59.8 (5.14)	64.0 (5.28)	65.3 (5.50)	65.4 (5.37)	65.4 (5.35)	65.4 (5.24)
	$M = 256$	20.4 (8.18)	60.3 (9.72)	64.4 (10.11)	65.6 (10.16)	65.8 (10.28)	65.8 (10.13)	65.8 (10.18)
$B = \$1$	$M = 16$	20.0 (1.24)	60.8 (2.33)	75.6 (2.76)	79.9 (2.92)	84.5 (3.07)	85.6 (3.21)	86.3 (3.21)
	$M = 32$	20.1 (2.21)	64.0 (3.85)	78.5 (5.02)	81.6 (5.36)	84.9 (5.63)	86.1 (5.72)	86.5 (5.81)
	$M = 64$	20.6 (3.85)	65.8 (7.29)	79.7 (9.35)	82.8 (9.88)	85.7 (10.54)	86.6 (10.95)	86.6 (11.10)
	$M = 128$	20.4 (7.24)	67.2 (14.07)	80.7 (17.90)	83.4 (19.35)	86.1 (20.18)	86.7 (20.81)	86.7 (21.49)
	$M = 256$	20.5 (14.19)	67.6 (27.65)	81.0 (36.63)	83.5 (38.10)	86.1 (39.72)	86.7 (42.68)	86.8 (43.11)
$B = \$20$	$M = 16$	20.0 (1.35)	60.9 (2.35)	77.2 (3.62)	81.8 (3.84)	89.9 (4.20)	92.8 (4.54)	94.7 (4.83)
	$M = 32$	20.1 (2.16)	64.0 (4.45)	79.9 (6.57)	83.5 (7.01)	91.4 (7.96)	93.9 (8.46)	95.9 (9.07)
	$M = 64$	20.6 (4.04)	65.8 (8.05)	81.0 (12.48)	84.7 (13.79)	92.5 (15.84)	95.0 (16.40)	96.4 (17.58)
	$M = 128$	20.4 (7.41)	67.2 (15.99)	82.0 (24.50)	85.6 (27.04)	92.8 (30.13)	95.9 (32.35)	96.8 (34.50)
	$M = 256$	20.5 (14.38)	67.6 (29.99)	82.4 (49.16)	86.1 (51.57)	93.2 (60.87)	96.2 (64.39)	96.9 (66.99)

D.3 Analysis of Monte Carlo Budgets

Table 2 demonstrates constrained completion probability and mean cumulative planning wall-clock time per complete ProofFlow execution under different Monte Carlo budgets. Performance improves from small M to moderate M but quickly saturates, while cumulative planning time continues to increase. For example, under $B = \$20$ and $D = 120$ minutes, increasing M from 64 to 256 improves constrained completion probability only from 96.4 to 96.9, but increasing mean cumulative planning time from 17.58 seconds to 66.99 seconds. Thus, $M = 64$ provides a strong quality-runtime trade-off across budget-deadline settings.

This saturation suggests that MCPP benefits primarily from online lookahead over downstream constrained completion, rather than sampling alone. Once enough samples are available to distinguish high-value actions, additional simulations mostly reduce estimator variance and provide limited gains. Therefore, although larger M can theoretically reduce the approximation gap as discussed in Appendix B.3, moderate Monte Carlo budgets are sufficient in practice. We use $M = 64$ as the default setting in the main experiments.

E Prompts

In this section, we present the full prompt templates used in our experiments.

CodeFlow Multi-Turn Input Serialization Template

```
System Message:
You are a Programming Expert. You always provide correct and reliable code solutions.

User Message:

You are a Programming Expert. You always provide correct and reliable code solutions. You will be provided
with the Background of the whole problem, a programming problem and may also some pre-implemented functions.
If pre-implemented functions provided, you need to call the pre-implemented functions and write a new
function to solve the problem.

Background of the whole problem:
{{ problem_description }}

Problem Description:
You need to complete {{ name }} function.
{{ statement }}

[Optional Dependency Block]
Dependency information:
To solve the problem, you need to utilize the pre-implemented functions {{ dependencies }} provided.

Pre-implemented functions:
{{ history }}

Guidelines:
- Ensure the function is executable and meets the requirement.
- Handle dependency information correctly when dependencies are provided.
- Provide clear and concise comments to explain key parts of the code.

[Final-Turn Prefix]
For the final subproblem, generate code beginning with:
import sys
def {{ name }}():
    input = sys.stdin.read().split()

Return your response by filling the function body following the function signature provided. Just generate
the function itself and do not output examples.

“‘python
```

Figure 7: **CodeFlow multi-turn input serialization template.** The current subproblem is serialized with the whole-problem background, the target function name, the subproblem statement, and optionally dependency names and previously generated implementations. The dependency and final-turn blocks are included according to the subproblem position in the workflow.

ProofFlow Formalizer Input Serialization Template

```
System Message:
You are a thinking model specialized in turning natural-language math statements into Lean 4 code.

User Message:
Please autoformalize the following natural language problem proof step in Lean 4.

Use the following lemma name:
{{ lemma_header }}

The natural language statement is:
{{ item.statement }}

The dependencies are:
{{ dependencies }}

This is the Lean code skeleton you need to use:

— lean4 —
import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat Filter
{{ lemma_header }}
[place correct hypothesis here] :
[place goal here] := by
sorry
— end lean4 —

[Optional Verified Dependency Context]
The following Lean 4 code contains verified declarations and proofs you may use:
{{ dependency_context_code }}

Important: Please write only one lemma or theorem.
```

Figure 8: **ProofFlow formalizer input serialization template.** Each proof-graph node is converted into a Lean formalization task using the node statement, dependency identifiers, and a fixed Lean skeleton. When available, verified prior Lean declarations are appended as dependency context.

ProofFlow Solver Input Serialization Template

```
System Message:
You are an expert Lean 4 theorem prover. Your job is to complete partially written Lean 4 code and provide
correct, verifiable proofs.

User Message:

This is the lemma/theorem I want you to prove:
{{ item.statement }}

Complete the following Lean 4 code. Do not remove imports:

“‘lean4
{{ item.formalization["lean_code"] }}
“‘

You can adapt previous Lean 4 lemma statements to fit the goal, especially if you encounter errors.

[Optional Verified Dependency Context]
The following Lean 4 code contains already verified previous proof steps and declarations that this node
may depend on:
{{ dependency_context_code }}

[Optional Error Notice]
The previous Lean 4 code contains errors. Please take that into account.
```

Figure 9: **ProofFlow solver input serialization template.** After formalization, the solver receives the target proof-step statement, the generated Lean skeleton, and optionally verified dependency code. The model is instructed to replace the placeholder proof with a complete Lean 4 proof without `sorry`.