JOVE: JOINT EXECUTION AND VERIFICATION FOR RESOURCE-AWARE LLM TASK GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Complex reasoning queries can be decomposed into directed acyclic task graphs and distributed across heterogeneous LLMs, reducing latency through parallelism and enabling smaller models to solve complex tasks. In practice, however, the suitability of an LLM for a given subtask may be a priori unknown, and execution alone does not reveal output correctness. We propose JOVE, an online framework that jointly assigns executor LLMs and selects intermediate outputs for paid verification. Verification runs asynchronously and is used to improve future allocations, so the system must balance spending on execution now against learning for later. We study how to optimize this trade-off under a long-term budget and a per-query latency constraint, with stochastic, initially unknown LLM service quality, invocation costs, and execution times. JOVE makes execution and verification decisions by solving a sequence of per-query mixed-integer linear programs. Online learning updates task-dependent estimates of LLM quality based on verification feedback, while an information-gain bonus incorporates the value of learning into allocation decisions. Under a natural set of assumptions, we establish sublinear quality-learning regret for JOVE. Across four reasoning benchmarks, JOVE achieves competitive accuracy against standard inference baselines while reducing average cost and latency by at least 3.17 times. Code: link.

1 INTRODUCTION

Large language models (LLMs) offer different combinations of capability, specialization, and serving cost. This heterogeneity has motivated *LLM routing*: instead of sending every query to the strongest model, a router might select a less expensive model when that is likely to suffice (Chen et al., 2023; Ong et al., 2025; Wang et al., 2025). Model selection is of particular interest for complex reasoning queries, which often benefit from an agentic workflow that decomposes the original problem into smaller, interdependent subtasks. A natural abstraction is *graph-based inference*: a planner decomposes a query into a directed acyclic graph (DAG) whose nodes represent subtasks and whose edges indicate dependencies between them. The node outputs are then used to construct the final response. This structure enables lightweight models to handle individual subtasks and reduces latency through parallel execution of independent branches.

Prior work spans three directions. (1) Graph-structured decoding organizes intermediate reasoning or exposes parallelism, typically with a fixed LLM (Ning et al., 2024; Jin et al., 2025a). (2) LLM routing selects models per query or through sequential interactions (Chen et al., 2023; Zhang et al., 2025). (3) Graph-aware LLM systems coordinate different model calls within task graphs (Kim et al., 2023; Dong et al., 2026a). Existing routing and graph-aware allocation methods typically learn from offline data or assume free model-quality feedback after each call. However, in practice, obtaining such feedback may require a judge model or human evaluator at an extra cost. **In this paper, we therefore ask:** given a query and its generated task graph, how should a resource-constrained system jointly select executor LLMs and decide which node outputs to verify, balancing current-query performance against paid verification that improves future allocations?

Challenges and our design. We study the problem under two practical resource constraints: a long-term average *API-cost budget* shared across queries, which offers the flexibility to absorb cost fluctuations over rounds, and *delivery deadlines* set per query according to its urgency and difficulty. (1) The first challenge is allocating heterogeneous LLMs under uncertain model quality, cost, and

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

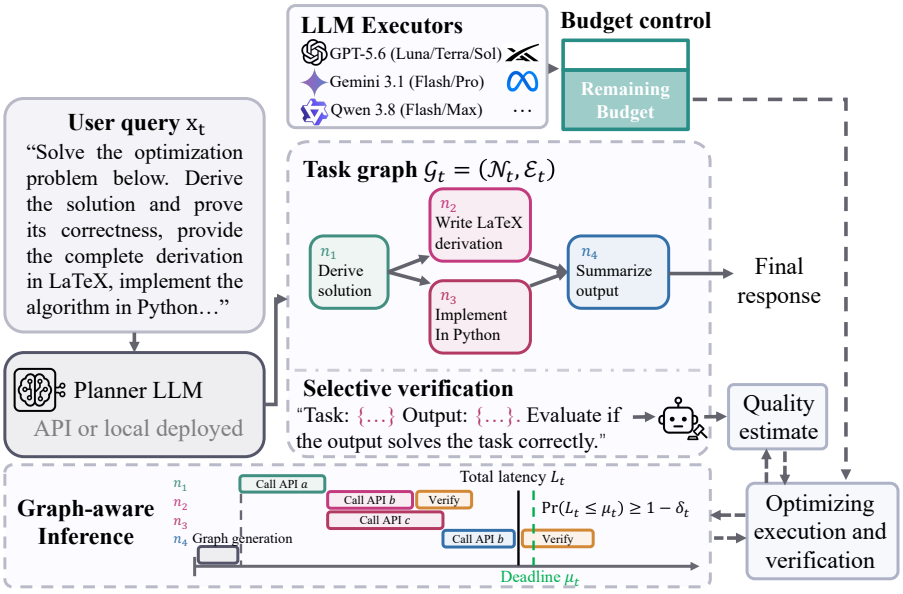


Figure 1: Overview of JOVE. A planner decomposes each query into a task graph and identifies candidate executor LLMs for each node. JOVE jointly assigns executors and selects outputs for verification, subject to a long-term execution/verification budget and latency constraints. Verification proceeds asynchronously without delaying the current response, improving model-quality estimates.

latency while satisfying both constraints. The budget limit couples decisions across queries: spending more on difficult queries requires saving on others. The deadline couples decisions within each task graph: choosing a slower executor at one node can restrict feasible choices downstream, even when it is locally favorable in quality and cost. (2) The second challenge is obtaining sufficiently granular model-quality feedback without wasting resources by verifying every output. A model call reveals its cost and latency but not output correctness, and the final response feedback may be too coarse to attribute errors to individual node-model assignments. We therefore introduce **selective node-level verification** to provide feedback on individual outputs at an extra cost. Each node contains a narrowly scoped subtask and an explicit success criterion. After a selected node completes, its subtask, outputs, and criterion are sent to a fixed verifier (a human evaluator or a judge model), which returns a binary correctness label. This label updates the estimated quality of the selected model for similar subtasks, informing future allocations. Verification is not required to construct the current response and can proceed asynchronously after node completion. We therefore assume that it does not delay response delivery, although its cost counts toward the long-term query cost.

In this paper, we propose **JOVE**, **J**oint **O**ptimization of **V**erification and **E**xecution, a framework that jointly selects executor LLMs and outputs to verify under system resource constraints (Figure 1). We formulate the problem as an online constrained optimization problem that maximizes average node correctness subject to a long-term average API-cost budget and a probabilistic latency limit for each query. JOVE makes allocation decisions by solving per-query mixed-integer linear programs (MILPs), using online estimates of model quality, cost, and latency, with an information-gain bonus that values verification feedback for future allocations. Our main contributions are:

- **Joint execution–verification formulation.** We formulate task-graph inference as an online problem of jointly choosing executor LLMs and which outputs to verify, with initially unknown model quality, cost, and latency, under a long-term cost budget and a per-query latency chance constraint.
- **Verification-aware online allocation.** We develop JOVE, a per-query MILP policy combining online quality estimates with an information-gain bonus for verification. We prove sublinear quality-learning regret when verification is frequent enough to learn but does not crowd out execution.
- **Empirical evaluation.** Across multiple reasoning datasets, JOVE achieves average accuracy comparable to the strongest baseline while achieving 3.7–16.3× lower cost and 8.3–17.0× lower latency. Ablation studies examine the effects of different verification weights and resource constraints.

2 Related Work

Graph-structured inference. Graph structure can improve reasoning or expose opportunities for parallel execution. Tree-of-Thoughts and Graph-of-Thoughts organize and search over intermediate reasoning states (Yao et al., 2023; Besta et al., 2024). Skeleton-of-Thought, PASTA, and Plato instead exploit independence or explicit dependencies to generate parts of a response concurrently (Ning et al., 2024; Jin et al., 2025b;a). HuggingGPT, LLMCompiler, and S-DAG extend workflow planning to collections of different models, tools, and subagents (Shen et al., 2023; Kim et al., 2023; Dong et al., 2026b). Dyserve further includes verifiers to gate model selection using static offline profiles (Qian et al., 2026). JOVE takes a planner-generated DAG as input and learns online to optimize executor assignments and which outputs to verify under resource constraints.

Resource-aware LLM routing and paid feedback. LLM routers and cascades improve quality-cost trade-offs by selecting models based on predicted quality, difficulty, or preference (Chen et al., 2023; Ding et al., 2024; Ong et al., 2025; Dekoninck et al., 2024; Feng et al., 2025). Recent online and resource-aware methods further consider feedback, budgets, or latency in their problems (Wang et al., 2025; Panda et al., 2025; Li et al., 2026; Patel et al., 2026; Zhang et al., 2025; Atalar et al., 2025), mostly assuming that quality feedback is freely available after execution. AutoMix verifies an initial output before routing, incurring verification cost and delay on every query rather than verifying selectively (Aggarwal et al., 2024). HybridFlow extends routing to task graphs, training its policy on offline data with subsequent online calibration, but restricts each routing decision to either an edge or a cloud model (Dong et al., 2026a). JOVE instead selectively verifies node outputs and learns an executor-allocation policy online. This selective-feedback setting relates to costly-observation bandits, which separate action selection from paying to observe its reward (Tucker et al., 2023; Schur et al., 2026). JOVE extends this perspective to DAGs by jointly selecting executors and outputs to verify under a shared execution-verification budget and per-query latency constraints.

3 Method

In this section, we formulate the joint execution and verification problem (Section 3.1) and derive a per-query oracle MILP assuming known executor statistics (Section 3.2). We then develop an online policy replacing these statistics with estimates and adding an information-gain bonus for verifications (Section 3.3). Finally, we establish a sublinear quality-learning regret guarantee (Section 3.4).

3.1 Problem formulation

Let x_t denote the t-th query, where $t = 1, 2, \ldots$ indexes query arrivals. A planner LLM decomposes x_t into a DAG $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{E}_t)$, where each node $i \in \mathcal{N}_t$ is a subtask. An edge $(j, i) \in \mathcal{E}_t$ indicates that node i requires an output from node j before execution. Each node specifies a subtask instruction, required upstream outputs, a success criterion, and a candidate executor set $\mathcal{A}_t^i$. The planner decides $\mathcal{A}_t^i$ based on the subtask's needs and each model's capabilities, supported modalities, and context length; thus, candidate sets may differ across nodes. A designated output node produces the final response. All executor and verification decisions are made before graph execution. Appendix A.6 summarizes the notation used throughout the paper.

Executor selection and verification. For node i of query x_t, an online policy π selects the joint action $(a_t^i, h_t^i) \in \mathcal{A}_t^i \times \{0, 1\}$, where a_t^i is the executor model and $h_t^i = 1$ selects its output for verification[1]. Executing node i with model a incurs stochastic API cost $M_t^i(a)$ and latency $D_t^i(a)$. Its latent correctness $Y_t^i(a) \in \{0, 1\}$ indicates whether the output satisfies the node's success criterion and is not observed from execution alone. If $h_t^i = 1$, a fixed verifier LLM (or human judge), shared across nodes, assesses the subtask, output, and criterion at cost $\widetilde{M}_t^i$. It returns a binary label $\widehat{Y}_t^i \in \{0, 1\}$ estimating $Y_t^i(a_t^i)$. Verification informs future allocations but does not alter the current response. We assume that verification proceeds asynchronously without delaying graph execution or response delivery. Its cost counts toward the API budget, whereas its latency is excluded from the current query's response time.

[1] We leave the dependence of a_t^i and h_t^i on π implicit for notational simplicity.

Graph-level cost and latency. Let $\mathbf{a}_t = \{a_t^i\}_{i\in\mathcal{N}_t}$ and $\mathbf{h}_t = \{h_t^i\}_{i\in\mathcal{N}_t}$. The total API cost of processing query x_t is a random variable:

$$C_t(\mathbf{a}_t, \mathbf{h}_t) = M_t^{\text{plan}} + \sum_{i\in\mathcal{N}_t} \left(M_t^i(a_t^i) + h_t^i \widetilde{M}_t^i \right), \tag{1}$$

where M_t^{plan} denotes the random API cost of task graph generation. A local planner incurs no API charge, so $M_t^{\text{plan}} = 0$. Devices with limited computational resources may instead use a remote LLM API planner. Its realized cost m_t^{plan} is observed before executor allocation and is therefore constant with respect to the allocation decisions. Let d_t^{plan} denote the task graph generation time plus the time required to allocate executor models. Assuming that each node starts as soon as its required upstream outputs are available, with no additional scheduling delays, graph execution latency equals the critical-path duration: $L_t = \max_{P\in\mathcal{P}(\mathcal{G}_t)} \sum_{i\in P} D_t^i(a_t^i)$, where $\mathcal{P}(\mathcal{G}_t)$ is the set of source-to-sink paths. Provided that executor allocation is completed within its reserved time, the end-to-end inference latency is therefore at most $d_t^{\text{plan}} + L_t$.

Objective and constraints. Let $\mathcal{H}_t$ denote the information available before allocating query x_t: past allocations with their realized costs, latencies, and verification labels, together with the current graph $\mathcal{G}_t$ and the realized planning cost and overhead. We maximize the long-run average fraction of correctly executed nodes per query, subject to a long-run average API-cost budget γ per query and a deadline μ_t with violation tolerance $\delta_t \in (0,1)$ for each query:

$$\max_{\pi} \quad \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_\pi \left[\frac{1}{|\mathcal{N}_t|} \sum_{i\in\mathcal{N}_t} Y_t^i(a_t^i) \right] \tag{2a}$$

$$\text{s.t.} \quad \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_\pi[C_t(\mathbf{a}_t, \mathbf{h}_t)] \le \gamma, \tag{2b}$$

$$\Pr_\pi\Big(L_t \le \mu_t - d_t^{\text{plan}} \,\Big|\, \mathcal{H}_t, \mathbf{a}_t \Big) \ge 1 - \delta_t, \quad \forall t,\ \forall \mathcal{H}_t,\ \forall \mathbf{a}_t \tag{2c}$$

$$a_t^i \in \mathcal{A}_t^i, \quad h_t^i \in \{0,1\}, \quad \forall i \in \mathcal{N}_t,\ \forall t. \tag{2d}$$

Interpretation. The objective is an additive proxy for the quality of the final-answer: it encourages reliable intermediate results, and, under a sound decomposition of the original query, correct execution of all subtasks is sufficient for a correct final response. The budget permits higher spending on difficult queries to be offset by savings on others. Latency is constrained per query because a fast response cannot compensate for another missing its deadline; conditioning Eq. 2c on the realized history and selected allocation enforces this at every round.

Remark 3.1 (Role of h_t^i). If expected model qualities were known, $h_t^i = 0$ would be optimal since verification adds cost without changing the response. When the qualities are unknown (Section 3.3), setting $h_t^i = 1$ provides a label $\widehat{Y}_t^i$ for learning model quality and informing future allocations.

3.2 Oracle MILP Formulation

Oracle coefficients and joint encoding. We first consider an oracle setting in which model quality, cost, and latency statistics are known. To encode executor and verification decisions jointly, define $z_t^i(a,\eta) = \mathbb{1}\{a_t^i = a,\ h_t^i = \eta\}$, where $\sum_{a\in\mathcal{A}_t^i} \sum_{\eta\in\{0,1\}} z_t^i(a,\eta) = 1$, $\forall i \in \mathcal{N}_t$. Define the expected execution and verification cost at node i as $\bar{c}_t^i(a,\eta) = \mathbb{E}\Big[M_t^i(a) + \eta \widetilde{M}_t^i \Big]$. The expected total cost is $m_t^{\text{plan}} + \sum_{i\in\mathcal{N}_t} \sum_{a\in\mathcal{A}_t^i} \sum_{\eta\in\{0,1\}} \bar{c}_t^i(a,\eta) z_t^i(a,\eta)$.

Long-term budget via dynamic pricing. We associate Eq. 2b with a nonnegative multiplier λ_t, interpreted as the current shadow price of API spending. Starting from $\lambda_1 = 0$, we update the price after observing query t's realized total cost c_t, including planning, execution, and verification:

$$\lambda_{t+1} = [\lambda_t + \alpha_\lambda (c_t - \gamma)]_+, \qquad \lambda_1 = 0, \tag{3}$$

where $\alpha_\lambda > 0$ is the step size and $[u]_+ = \max\{u, 0\}$. Spending above γ increases the penalty on API costs, whereas spending below decreases it, subject to $\lambda_t \ge 0$. Let $\bar{y}_t^i(a) = \mathbb{E}[Y_t^i(a)]$. For

query t, the price computed from preceding queries enters the objective as

$$\max_{\mathbf{z}_t} \sum_{i\in\mathcal{N}_t} \sum_{a\in\mathcal{A}_t^i} \sum_{\eta\in\{0,1\}} \left[\frac{1}{|\mathcal{N}_t|}\bar{y}_t^i(a) - \lambda_t \bar{c}_t^i(a,\eta)\right] z_t^i(a,\eta). \tag{4}$$

Appendix A.2.2 establishes a long-term budget guarantee for the oracle policy under the stated environment and feasibility assumptions.

Per-query latency via node-level quantiles. To enforce Eq. 2c, we split the violation tolerance δ_t across all execution nodes, choosing $\delta_t^i > 0$ with $\sum_{i\in\mathcal{N}_t} \delta_t^i \le \delta_t$. For each node and candidate executor, define the smallest deadline met with probability at least $1-\delta_t^i$ as

$$\bar{d}_t^i(a) = \ell_t^i(a,\delta_t^i) = \inf\left\{d \ge 0 : \Pr\left(D_t^i(a) \le d | \mathcal{H}_t\right) \ge 1-\delta_t^i\right\}. \tag{5}$$

A union bound then gives the following sufficient condition for Eq. 2c:

$$\max_{P\in\mathcal{P}(\mathcal{G}_t)} \sum_{i\in P} \sum_{a\in\mathcal{A}_t^i} \bar{d}_t^i(a) \sum_{\eta\in\{0,1\}} z_t^i(a,\eta) \le \mu_t - d_t^{\text{plan}} \Rightarrow \Pr\Big(L_t \le \mu_t - d_t^{\text{plan}} | \mathcal{H}_t, \mathbf{a}_t\Big) \ge 1-\delta_t. \tag{6}$$

Appendix A.2.1 provides the proof. This reduction may be conservative, but requires no independence among node latencies. We use $\delta_t^i = \delta_t/|\mathcal{N}_t|$ by default. Appendix A.5.1 considers a less restrictive allocation that jointly selects executors and node risk δ_t^i from a discrete grid of risk levels.

Oracle MILP. Equation 4 adds a cost penalty to the original objective (Eq. 2a), yielding a linear objective in the allocation variables. To formulate the oracle MILP, we express the deterministic latency condition in Eq. 6 as a set of linear constraints, by introducing finish-time variables F_t^i. Let $\mathcal{S}_t$ and $\mathcal{T}_t$ denote the sets of source and sink nodes, respectively. Define $\mathbf{z}_t^i = \{z_t^i(a,\eta)\}_{a\in\mathcal{A}_t^i,\eta\in\{0,1\}}$ and let $v_t^i(\mathbf{z}_t^i) = \sum_{a\in\mathcal{A}_t^i} \bar{d}_t^i(a) \sum_{\eta\in\{0,1\}} z_t^i(a,\eta)$ be the selected latency quantile for node i. The deterministic path-length condition (Eq. 6) holds if and only if the following inequalities are feasible:

$$F_t^i \ge v_t^i(z), i \in \mathcal{S}_t, \quad F_t^i \ge F_t^j + v_t^i(z), (j,i) \in \mathcal{E}_t, \quad F_t^i \le \mu_t - d_t^{\text{plan}}, i \in \mathcal{T}_t. \tag{7}$$

These inequalities account for task dependencies and require every path to finish by the deadline; together with the binary assignment constraints and Eq. 4, they define a per-query MILP. The complete formulation is given in Appendix A.1. For later use, let $\mathbf{d}_t = \{\bar{d}_t^i(a)\}_{i,a}$ and let $\mathcal{F}_t(\mathbf{d}_t)$ denote the resulting feasible set of assignment and finish-time variables. Since the oracle already knows model qualities, verification provides no learning benefits; thus, choosing $\eta = 0$ at every node is optimal. We retain η to accommodate verification when model qualities are unknown (Section 3.3).

3.3 Online estimation and verification-aware allocation

The oracle optimization requires expected model correctness $\bar{y}_t^i(a)$, expected execution and verification cost $\bar{c}_t^i(a,\eta)$, and latency quantiles $\bar{d}_t^i(a)$, which are unknown in the online setting. Execution directly reveals cost and latency, whereas selective verification provides correctness labels for a subset of nodes. We use these observations to estimate the coefficients in an online fashion and augment the MILP objective (Eq. 4) with an information-gain bonus for acquiring verification feedback.

Model quality estimation. For each task-model pair, we embed the task instruction, descriptions of required node outputs, and candidate model description into a feature vector $\phi_t^i(a) \in \mathbb{R}^d$ using the embedding model specified in Appendix A.3.3. Let $(\phi, \hat{y}) \in \mathcal{D}_{t-1}$ denote the feature–label pairs obtained through verification before query t. Ridge regression with regularization $\lambda_{\text{reg}} > 0$ yields

$$G_{t-1} = \lambda_{\text{reg}} I + \sum_{(\phi,\hat{y})\in\mathcal{D}_{t-1}} \phi\phi^\top, \qquad \hat{\theta}_t = G_{t-1}^{-1} \sum_{(\phi,\hat{y})\in\mathcal{D}_{t-1}} \phi\hat{y}. \tag{8}$$

Using a linear upper confidence bound (LinUCB) construction (Abbasi-Yadkori et al., 2011), we define the uncertainty score and model quality estimate as

$$u_t^i(a) = \sqrt{\phi_t^i(a)^\top G_{t-1}^{-1} \phi_t^i(a)}, \quad \text{UCB}_t^i(a) = \text{clip}(\phi_t^i(a)^\top \hat{\theta}_t + \beta_t u_t^i(a), 0, 1), \tag{9}$$

where β_t scales the uncertainty bonus and $\text{clip}(\cdot, 0, 1)$ restricts the estimate to $[0, 1]$. Past verified observations reduce uncertainty along the feature directions they cover, supporting more accurate

quality predictions. $\beta_t u_t^i(a)$ favors underexplored task–model pairs. Appendix A.2.3 specifies β_t and establishes the high-probability confidence bound used in the regret analysis.

Resource estimation. Unlike correctness, cost and latency are observed directly from each executor call, so their estimates can be updated without additional feedback cost. The UCB score encourages selecting underexplored task-model pairs, helping resource estimates for each executor a cover a broader range of input contexts. We reuse the task-model features ϕ to estimate expected costs and latency quantiles, weighting past observations by their similarity to the current feature vector $\phi_t^i(a)$. This yields the estimated execution-plus-verification cost $\hat{c}_t^i(a,\eta)$ and node-level latency quantile $\hat{d}_t^i(a)$ used in the online MILP. Appendix A.3.4 provides the estimation details.

Verification-aware allocation. We replace the unknown coefficients in the oracle MILP (Section 3.2) with the above estimates. Execution reveals cost and latency, but model quality feedback requires paid verification. Since verification does not alter the current response, we reward its information gain to support future executor selection. Specifically, verifying a task-model pair (i,a) updates the information matrix to $G_{t-1}^+ = G_{t-1} + \phi_t^i(a)(\phi_t^i(a))^\top$. We quantify the information gain from this observation using the D-optimal criterion (de Aguiar et al., 1995),

$$I_t^i(a) = \frac{1}{2}\log\frac{\det(G_{t-1}^+)}{\det(G_{t-1})} = \frac{1}{2}\log\big(1+u_t^i(a)^2\big)\,, \tag{10}$$

Interpretation. The eigenvalues of G_{t-1} measure accumulated information, including regularization, along its eigenvector directions. Since the determinant is the product of these eigenvalues, the gain $I_t^i(a)$ is large when the candidate feature vector has a substantial component in a direction with limited prior observations, and small when those representations are well represented. For a fixed confidence radius β_t, this gain equals the reduction in the log-volume of the parameter confidence ellipsoid after adding the observation. For the online MILP, JOVE replaces the oracle coefficients with online estimates and adds a verification bonus:

$$\max_{(\mathbf{z},\mathbf{F})\in\mathcal{F}_t(\hat{\mathbf{d}}_t)} \sum_{i\in\mathcal{N}_t}\sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}}\left[\frac{1}{|\mathcal{N}_t|}(\mathrm{UCB}_t^i(a)+k_v\eta I_t^i(a)) - \lambda_t\hat{c}_t^i(a,\eta)\right] z_t^i(a,\eta). \tag{11}$$

The three terms represent estimated model quality, verification gain, and priced API cost, respectively. The feasible set $\mathcal{F}_t(\hat{\mathbf{d}}_t)$, defined in Section 3.2, enforces assignment constraints and the latency condition using estimated node durations. The budget price λ_t adapts to observed spending, while $k_v \geq 0$ controls the weights assigned to verification gain. After each query, we update the quality model using verified labels (Eq. 8), the resource estimates using observed calls (Eq. 21 and 22), and λ_t using the realized total cost (Eq. 3). Appendix A.3.1 presents the complete JOVE loop.

Remark 3.2 (Verification as a threshold decision). For a fixed executor, verification leaves latency feasibility and quality term unchanged. It is therefore optimal to set $h_t^i = \mathbf{1}\{k_v I_t^i(a_t^i)/|\mathcal{N}_t| \geq \lambda_t \hat{\widetilde{m}}_t^i\}$. Optimizing over h_t^i is equivalent to adding $[k_v I_t^i(a)/|\mathcal{N}_t| - \lambda_t \hat{\widetilde{m}}_t^i]_+$ to each candidate executor's score, so the value of verification also influences executor selection. JOVE can prefer an executor offering valuable future feedback over one with a higher immediate quality score.

3.4 Regret Analysis

In this section, we establish sublinear quality-learning regret for a range of verification weights k_v under the following assumptions.

Assumption 3.3 (Quality and feedback)**.** There exists a fixed parameter $\theta^\star$ such that, for every available task-model pair, $\bar{y}_t^i(a) = \phi_t^i(a)^\top\theta^\star \in [0,1]$, where $\|\phi_t^i(a)\|_2 \leq L$ and $\|\theta^\star\|_2 \leq S$. Verified labels satisfy $\widehat{Y}_t^i = \bar{y}_t^i(a_t^i) + \varepsilon_t^i$. Conditioned on the selected features and all preceding observations, ε_t^i has zero-mean and is R-sub-Gaussian. Selected verification labels and realized query costs are available before the next query is allocated.

Assumption 3.4 (Allocation regularity)**.** For all $t \leq T$, $1 \leq |\mathcal{N}_t| \leq n_{\max}$, $0 \leq \lambda_t \leq \lambda_{\max}$, $0 \leq \hat{\widetilde{m}}_t^i \leq \widetilde{m}_{\max}$, and $\lambda_{\mathrm{reg}} \geq L^2$. For every $t \leq T$, the estimated feasible set $\mathcal{F}_t(\hat{\mathbf{d}}_t)$ is nonempty, and the online MILP (Eq. 11) is solved to optimality.

Assumption 3.3 formalizes a stationary quality model and unbiased feedback. Assumption 3.4 bounds priced verification costs and requires optimal feasible allocations. Under these conditions, we compare JOVE with the following benchmark policy.

Benchmark policy. To isolate quality learning, we compare JOVE with a policy that knows the true expected qualities $\bar{y}_t^i(a)$ but uses the same resource estimates and budget prices as JOVE. We define

$$J_t(\mathbf{z}_t) = \sum_{i\in\mathcal{N}_t}\sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}}\big[\frac{1}{|\mathcal{N}_t|}\bar{y}_t^i(a) - \lambda_t\hat{c}_t^i(a,\eta)\big]z_t^i(a,\eta), \quad (\mathbf{z}_t^\star, \mathbf{F}_t^\star) \in \underset{(\mathbf{z},\mathbf{F})\in\mathcal{F}_t(\hat{\mathbf{d}}_t)}{\arg\max}\ J_t(\mathbf{z}). \quad (12)$$

The quality-learning regret is $\mathcal{R}_T = \sum_{t=1}^T [J_t(\mathbf{z}_t^\star) - J_t(\mathbf{z}_t)]$, where $\mathbf{z}_t$ is JOVE's selected allocation.

Theorem 3.5 (A sufficient verification-weight interval)**.** *Under Assumptions 3.3-3.4, fix $\delta_{\text{conf}} \in (0,1)$ and a horizon $T \geq 8d\log(8n_{\max})$, where d is the dimension of the feature vector $\phi_t^i(a) \in \mathbb{R}^d$. Let β_t be specified in Appendix A.2.3, and define $c_v = \lambda_{\max}\widetilde{m}_{\max} > 0$, $C_{\text{bat}} = \Big(1 + \frac{(n_{\max}-1)L^2}{\lambda_{\text{reg}}}\Big)\Big(1 + \frac{L^2}{\lambda_{\text{reg}}}\Big)$, and $\Gamma_T = C_{\text{bat}} d\log\Big(1 + \frac{n_{\max}TL^2}{d\lambda_{\text{reg}}}\Big)$. For any fixed $k_v \in [K_T, 2K_T]$, where $K_T = \frac{2n_{\max}c_v}{\log\big(1+(\Gamma_T/(n_{\max}T))^{2/3}\big)}$, with probability at least $1-\delta_{\text{conf}}$,*

$$\mathcal{R}_T \leq 4(n_{\max}c_v + \max_{t\leq T}\beta_t)(n_{\max}T)^{2/3}\Gamma_T^{1/3}. \quad (13)$$

For fixed d, δ_{conf}, $n_{\max}$, and the remaining assumption constants, this bound is $O(T^{2/3}[\log(1+T)]^{5/6})$, which is sublinear in T (see Appendix A.2.3 for proof).

Interpretation. The theorem identifies an interval of verification weights yielding sublinear quality-learning regret under the stated assumptions. The lower bound K_T ensures verification of sufficiently uncertain task-model pairs, while the upper bound $2K_T$ keeps the cumulative bonus contribution sublinear in T. This horizon-dependent interval is sufficient for sublinear regret, but does not identify an optimal k_v. The condition $T \geq 8d\log(8n_{\max})$ is conservative. Reducing d lowers this threshold and may enable faster learning, but also restricts the quality model's expressiveness, potentially reducing prediction accuracy. Empirically, JOVE performs well across a broad range of verification weights, suggesting limited sensitivity to k_v within the tested range.

4 Experiments

Datasets and metrics. We evaluate JOVE on four reasoning benchmarks: Bamboogle (Press et al., 2023) (multi-hop QA), MMLU-Pro (Wang et al., 2024) (multidisciplinary reasoning), GPQA (Rein et al., 2023) (science), and LiveBench-Reasoning (White et al., 2025) (structured reasoning). We repeat each experiment over three random orderings of the online query sequence, and measure final answer accuracy, end-to-end response latency, and total API cost per query.

Experimental settings. We use Gemini-2.5-Flash-Lite as the planner and Qwen3-235B-A22B-2507 as the verifier. The executor pool includes eight models of different sizes (detailed in Appendix A.4), accessed through the OpenRouter platform. Unless otherwise specified, we set the verification weight to $k_v = 5.0$, exploration coefficient to $\beta_t = 0.25$, regularization parameter to $\lambda_{\text{reg}} = 1$, budget-update step size to $\alpha_\lambda = 100$, and latency violation tolerance to $\delta_t = 0.1$. We use Qwen3-32B as the reference model for accuracy, cost, and latency comparisons. Based on its average cost and latency on Bamboogle, we set the API-cost budget to $\gamma = 2.0 \times 10^{-4}$ USD and the deadline to $\mu_t = 16$s. We choose Bamboogle because it yields the lowest cost and latency across the four datasets, resulting in stricter resource targets than calibration across all datasets.

Baselines. We compare against two groups of methods. (1) *Inference-level baselines*: Direct invocation, Chain-of-Thought (CoT) (Wei et al., 2022), Skeleton-of-Thought (SoT) (Ning et al., 2024), and Plato (Jin et al., 2025a). We evaluate these methods using the reference model without budget and latency constraints. (2) *Resource-aware baselines*: *WR-Online* (Li et al., 2026) routes queries under the cost budget (without latency constraints) by learning cost and reward online as a constrained contextual bandit. *Knapsack-ascend* selects an allocation that fully utilizes the cost budget γ and latency allowance, while ensuring nondecreasing expected execution costs along each source-to-sink path. *Knapsack-descend* follows the same procedure but instead requires nonincreasing costs. As an ablation, we evaluate *JOVE single-node*, which retains JOVE's allocation and verification mechanisms but answers each query with one executor call, without decomposition. For resource-aware methods, we generate four distinct task graphs per query with 2–5 nodes and shuffle all query–graph pairs into an online learning sequence. We further discuss the learning sequence in Appendix A.4.1. We did not include resource-aware methods that require offline data (Dong et al., 2026a; Qian et al., 2026), since our setting provides no such data and the policy is learned online.

Table 1: Accuracy, latency, and API cost per query across four benchmarks. Averaged across benchmarks, JOVE achieves competitive accuracy with substantially lower latency and API cost than the unconstrained inference baselines. Values are mean ± standard deviation over three random seeds.

(a) Accuracy (%) ↑

Methods	Bamboogle	MMLU-Pro	LiveBench-Reasoning	GPQA
Inference-level baselines (unconstrained)				
Direct (Qwen3-32B)	47.5 ± 3.2	47.3 ± 0.8	31.4 ± 1.2	45.1 ± 3.8
CoT (Wei et al., 2022)	**51.7** ± 1.2	54.1 ± 1.1	39.2 ± 0.9	54.2 ± 3.6
SoT (Ning et al., 2024)	48.9 ± 3.4	52.0 ± 3.1	33.4 ± 2.3	50.9 ± 6.8
Plato (Jin et al., 2025a)	46.5 ± 0.9	52.8 ± 3.2	38.4 ± 1.8	**60.3** ± 0.1
Resource-aware methods (constrained)				
WR-Online (Li et al., 2026)	43.3 ± 3.5	45.7 ± 3.8	38.0 ± 4.6	48.1 ± 1.7
Knapsack-ascend	48.0 ± 2.1	41.0 ± 6.0	32.2 ± 5.6	39.0 ± 1.3
Knapsack-descend	44.3 ± 7.0	42.6 ± 2.9	33.2 ± 2.0	41.2 ± 1.2
JOVE single-node	28.8 ± 8.4	46.9 ± 11.4	28.1 ± 4.2	43.5 ± 9.4
JOVE	49.3 ± 1.0	**54.8** ± 1.1	**41.3** ± 1.2	52.7 ± 2.8

(b) Latency (sec.) and cost ($\times 10^{-4}$USD) ↓

Methods	Latency	Cost	Latency	Cost	Latency	Cost	Latency	Cost
Inference-level baselines (unconstrained)								
Direct (Qwen3-32B)	14.09 ± 0.99	2.00 ± 0.15	45.39 ± 2.29	5.34 ± 0.38	41.18 ± 0.20	10.54 ± 0.01	59.46 ± 10.64	10.56 ± 2.28
CoT (Wei et al., 2022)	18.61 ± 2.27	2.24 ± 0.41	34.11 ± 4.63	5.38 ± 0.34	80.38 ± 17.48	12.28 ± 1.31	101.11 ± 14.00	12.81 ± 1.96
SoT (Ning et al., 2024)	17.04 ± 9.71	10.56 ± 0.25	70.17 ± 6.27	27.53 ± 3.71	129.28 ± 23.46	52.62 ± 0.02	137.56 ± 37.40	55.46 ± 4.72
Plato (Jin et al., 2025a)	43.95 ± 4.62	8.89 ± 0.40	104.20 ± 5.21	22.23 ± 2.02	189.51 ± 41.48	40.17 ± 1.22	145.7 ± 36.94	37.35 ± 6.57
Resource-aware methods (constrained)								
WR-Online (Li et al., 2026)	16.02 ± 2.97	1.96 ± 0.10	21.17 ± 1.29	2.07 ± 0.02	23.08 ± 6.74	2.04 ± 0.13	17.25 ± 0.28	2.09 ± 0.09
Knapsack-ascend	4.66 ± 0.20	1.82 ± 0.03	4.70 ± 0.79	1.63 ± 0.25	16.68 ± 6.85	2.87 ± 1.18	5.65 ± 0.63	2.34 ± 0.40
Knapsack-descend	6.17 ± 0.69	1.97 ± 0.44	5.41 ± 1.17	1.72 ± 0.11	16.56 ± 6.84	2.37 ± 1.30	6.84 ± 1.36	1.88 ± 0.25
JOVE single-node	2.53 ± 1.10	0.66 ± 0.12	2.99 ± 0.43	1.60 ± 0.14	1.15 ± 0.21	2.30 ± 0.23	1.66 ± 0.60	2.15 ± 0.10
JOVE	3.68 ± 0.48	2.18 ± 0.18	6.66 ± 2.09	2.03 ± 0.16	8.71 ± 1.15	2.26 ± 0.43	9.32 ± 0.70	2.49 ± 0.14

4.1 Experimental results

JOVE performance. Table 1 reports accuracy, latency, and cost under the budget and latency deadline calibrated from direct Qwen3-32B calls on Bamboogle. Averaged over the four benchmarks, JOVE achieves comparable accuracy (49.5%) at 2.24×10^{-4} USD and 7.09s per query. Compared with the three inference baselines (CoT, SoT, and Plato), JOVE uses 3.7-16.3× less cost and 8.3-17.0× less latency while achieving comparable average accuracy. WR-Online achieves 43.8% average accuracy but does not enforce a latency constraint; its average latency is 2.7× that of JOVE. The constrained heuristics isolate the contribution of the allocation itself: Knapsack-ascend and -descend consume comparable resources yet reach only 40.1% and 40.3% on average, 9.5 and 9.2 points below JOVE. Averaged across benchmarks, JOVE single-node is cheapest and fastest (1.68×10^{-4} USD, 2.08s) but least accurate (36.8%); it has no task decomposition and receives at most one verification label per query, potentially limiting accuracy and slowing policy learning. JOVE uses selective node-level verification to learn the model-selection policy and allocates more capable executors where their estimated benefit justifies resource cost.

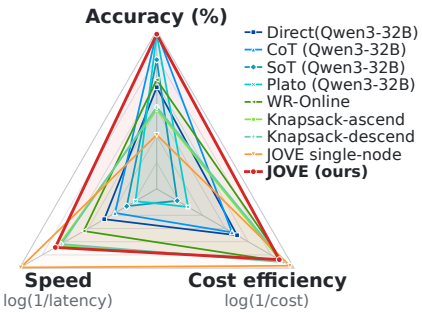
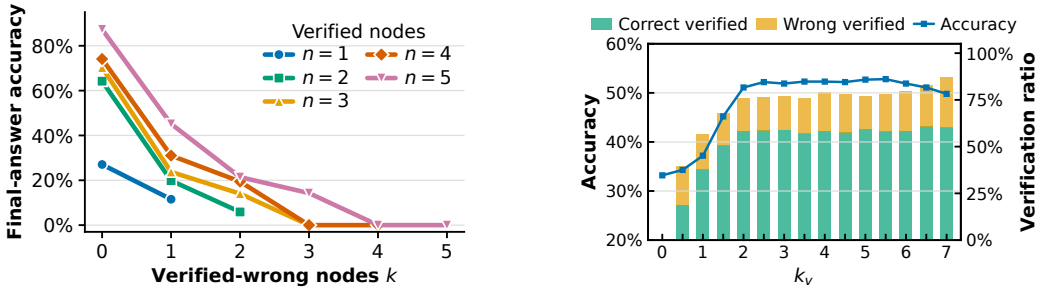


Figure 2: Accuracy, latency, and cost trade-offs, averaged over four datasets.

Figure 2 visualizes the accuracy-latency-cost trade-off reported in Table 1 using the average metrics over the four datasets. The inference-level baselines generally achieve promising accuracy at substantially greater latency and cost than JOVE. The constrained heuristics use comparable resources to JOVE but achieve lower average accuracy, highlighting the value of learning task-dependent executor quality. JOVE achieves a favorable balance across the three metrics by jointly selecting executors and verification decisions under the resource-aware objective.



(a) Node-level vs. final-answer correctness

(b) Effect of the verification weight k_v

Figure 3: Verification analysis of JOVE. (a) Final-answer accuracy versus the number of verifier-labeled incorrect nodes k, grouped by the number of verified nodes n, with results pooled across all four datasets. (b) GPQA final-answer accuracy and the fraction of executed nodes verified, split by verifier-labeled correct and incorrect output, as the verification weight k_v varies.

Node-level correctness vs. final-answer correctness. JOVE learns entirely from node-level verification without direct feedback on final response quality, maximizing average node correctness (Eq. 2a). Figure 3a shows how this additive surrogate relates to overall accuracy by grouping queries by total verified nodes $n \in \{1, \ldots, 5\}$ and verified-incorrect nodes k. The average final-answer accuracy drops monotonically as k increases for every n, indicating that node-level errors are associated with lower final-answer accuracy. At fixed k, accuracy is generally higher for larger n. When all five verified nodes are labeled correct $(n = 5, k = 0)$, final-answer accuracy reaches 87.3% rather than 100%. This gap may reflect verifier noise or imperfect alignment between node-level success criteria and final-answer correctness.

Selection of verification weights k_v. Figure 3b varies k_v on GPQA, reporting final-answer accuracy together with the fraction of executed nodes that are verified, split by whether the verifier judged them correct or incorrect. At $k_v < 1.5$, verification is infrequent and accuracy remains relatively low. A small k_v makes the weighted information gain less likely to outweigh the priced verification cost, limiting the feedback available for quality learning. For $k_v \in [2, 6]$, accuracy improves and remains stable, indicating that sufficient feedback is available and performance is relatively insensitive to the precise verification weight. With higher $k_v > 6$, accuracy starts to decline, potentially because verification consumes a growing share of the budget, leaving fewer resources for execution. This regime also shows an increasing fraction of executed nodes that are verified and labeled incorrect.

Satisfaction of the constraints. In Section 3.1, we introduce a long-term API-cost budget and a per-query latency chance constraint. In practice, JOVE replaces the initially unknown model statistics with online estimates, so the oracle guarantees do not directly extend to the practical policy. Constraint satisfaction then depends on resource-estimation accuracy and budget-price adaptation. Appendix A.4.2 evaluates this behavior empirically: as learning progresses, interval-average costs decrease toward or below the budget, while deadline hit rates improve, reaching at least 96% on all four datasets in the final interval against a 90% target.

The appendix provides additional results on the effect of the online learning sequence (Appendix A.4.1), ablations of cost budgets and latency deadlines (Appendix A.4.3), and comparisons of verification strategies (Appendix A.4.4).

5 Conclusion

We introduce JOVE, an online framework that optimizes the execution of complex reasoning queries represented by task graphs. By jointly managing LLM routing and selective paid verification via per-query MILP, JOVE balances immediate accuracy with feedback acquisition for future allocation decisions. We establish sublinear quality-learning regret under the stated assumptions and demonstrate effective budget and deadline control empirically. JOVE achieves competitive accuracy while substantially reducing cost and latency relative to baselines, providing a promising foundation for deploying efficient, multi-step LLM systems in resource-constrained environments.

AI USE STATEMENT

We used generative AI tools to help revise and polish the manuscript and to assist in writing code for figures and experimental baselines. The authors reviewed the AI-assisted text and code and take full responsibility for the manuscript, figures, implementations, and reported results.

ETHICS STATEMENT

We evaluate JOVE on publicly available question-answering and reasoning benchmarks. Our experiments involve no human participants or private user data, and we neither collect nor release a new dataset. JOVE inherits potential errors and biases from its executor and verifier models; verification feedback does not guarantee the correctness or safety of final responses. Deployment in high-stakes settings therefore requires application-specific validation and appropriate human oversight.

REPRODUCIBILITY STATEMENT

Section 3 describes JOVE's optimization and learning procedure, and Section 4 details the datasets, models, baselines, evaluation metrics, and experimental settings. Appendix A.1 presents the complete MILP formulation, Appendix A.2 provides the theoretical assumptions and proofs, and Appendices A.3–A.4 document the algorithm, prompts, resource estimators, and additional experimental details. The accompanying anonymous code repository contains implementations and scripts for reproducing the experiments and figures. Results may vary across runs because of stochastic model outputs and changes in external API providers' pricing and service conditions.

REFERENCES

Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24, 2011.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034, 2024.

Baran Atalar, Eddie Zhang, and Carlee Joe-Wong. Neural bandit based optimal llm selection for a pipeline of subtasks. *arXiv preprint arXiv:2508.09958*, 2025.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

P Fernandes de Aguiar, B Bourguignon, MS Khots, DL Massart, and R Phan-Than-Luu. D-optimal designs. *Chemometrics and intelligent laboratory systems*, 30(2):199–210, 1995.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for llms. *arXiv preprint arXiv:2410.10347*, 2024.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

Jiangwen Dong, Jiayu Li, Tianhang Zheng, and Wanyu Lin. Hybridflow: Resource-adaptive subtask routing for efficient edge-cloud LLM inference. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=erxWUWJlt2.

Jiangwen Dong, Zehui Lin, Wanyu Lin, and Mingjin Zhang. S-dag: A subject-based directed acyclic graph for multi-agent heterogeneous reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29394–29402, 2026b.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, volume 2025, pp. 26186–26203, 2025.

Shuowei Jin, Xueshen Liu, Yongji Wu, Haizhong Zheng, Qingzhao Zhang, Atul Prakash, Matthew Lentz, Danyang Zhuo, Feng Qian, and Z. Morley Mao. Plato: Plan to efficiently decode for large language model inference. In *Conference on Language Modeling*, 2025a.

Tian Jin, Ellie Y Cheng, Zack Ankner, Nikunj Saunshi, Blake M Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. *arXiv preprint arXiv:2502.11517*, 2025b.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*, 2023.

Yifei Li, Zihui Gao, and Laks VS Lakshmanan. Wiserouter: Llm routing with workload budget constraint. *arXiv preprint arXiv:2607.23765*, 2026.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting LLMs for efficient parallel generation. In *International Conference on Learning Representations*, 2024.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

OpenRouter. URL `https://openrouter.ai/`.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. Adaptive llm routing under budget constraints. In *EMNLP (Findings)*, pp. 23934–23949, 2025.

Shivam Patel, Akaash R Parthasarathy, Ankur Mallick, and Gauri Joshi. Beyond accuracy and cost: Latency-aware llm query routing for dynamic workloads. *arXiv preprint arXiv:2607.18253*, 2026.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Jiayi Qian, Yichong Zhang, Hanchen Yang, Chun Tao, Souvik Kundu, Zishen Wan, and Tushar Krishna. Serving agentic workflows with a physical-plan compiler and adaptive runtime, 2026. URL `https://arxiv.org/abs/2607.02942`.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

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. *arXiv preprint arXiv:2311.12022*, 2023.

Felix Schur, Jesus Lago, and Tanner Fiez. The price of knowledge: Optimal algorithms for costly bandits. In *Forty-Second Annual Conference on Uncertainty in Artificial Intelligence*, 2026.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Aaron D Tucker, Caleb Biddulph, Claire Wang, and Thorsten Joachims. Bandits with costly reward observations. In *Uncertainty in Artificial Intelligence*, pp. 2147–2156. PMLR, 2023.

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)*, pp. 10912–10922, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, volume 2025, pp. 91595–91631, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-R1: Teaching LLMs multi-round routing and aggregation via reinforcement learning. In *Advances in Neural Information Processing Systems*, 2025.

A APPENDIX

A.1 Oracle MILP formulation 13
A.2 Theoretical Analysis 14
A.2.1 Sufficient condition of the latency constraint 14
A.2.2 Average-budget guarantee 14
A.2.3 LLM quality learning analysis 16
A.3 Implementation details 23
A.3.1 JOVE procedure 23
A.3.2 Planner and verification prompts 24
A.3.3 Embedding model 24
A.3.4 Resource estimators 24
A.4 Additional experimental Details 25
A.4.1 Discussion on the learning sequence 26
A.4.2 Satisfaction of the constraints 26
A.4.3 Additional ablations 28
A.4.4 Comparison with random verification 28
A.5 Enhanced JOVE 29
A.5.1 Joint optimization of node risk allocation 29
A.6 Notation 31

A.1 ORACLE MILP FORMULATION

Let $\mathcal{S}_t$ and $\mathcal{T}_t$ denote the source and sink nodes of $\mathcal{G}_t$, respectively, and let F_t^i be the finish time of node i relative to the start of graph execution. Combining the priced objective with the deterministic latency reduction gives

$$
\begin{aligned}
&\max_{\mathbf{z}_t, \{F_t^i\}_{i\in\mathcal{N}_t}} \quad \sum_{i\in\mathcal{N}_t}\sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}} \left[\frac{1}{|\mathcal{N}_t|}\bar{y}_t^i(a) - \lambda_t \bar{c}_t^i(a,\eta)\right] z_t^i(a,\eta) \\
\text{s.t.} \quad & \sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}} z_t^i(a,\eta) = 1, && \forall i \in \mathcal{N}_t, \\
& z_t^i(a,\eta) \in \{0,1\}, && \forall i, a, \eta, \\
& F_t^i \geq \sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}} \bar{d}_t^i(a) z_t^i(a,\eta), && \forall i \in \mathcal{S}_t, \\
& F_t^i \geq F_t^j + \sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}} \bar{d}_t^i(a) z_t^i(a,\eta), && \forall (j,i) \in \mathcal{E}_t, \\
& F_t^i \leq \mu_t - d_t^{\text{plan}}, && \forall i \in \mathcal{T}_t, \\
& F_t^i \geq 0, && \forall i \in \mathcal{N}_t.
\end{aligned}
\tag{14}
$$

In the oracle MILP formulation, we assume the LLM expected performance $\bar{y}_t^i(a)$, expected invocation cost $\bar{c}_t^i(a,\eta)$ and node-level latency quantiles $\bar{d}_t^i(a)$ are all known, optimizing the executor selection and verification nodes using $z_t^i(a,\eta)$, and per-node finish time using F_t^i. In this oracle setting, verification delivers no gain on improving the estimation of LLM API qualities, it only increases the cost as defined in $\bar{c}_t^i(a,\eta)$.

A.2 Theoretical Analysis

Throughout, uppercase letters denote random quantities and lowercase letters their realizations.

A.2.1 Sufficient condition of the latency constraint

Lemma A.1 (Sufficient condition of the latency constraint)**.** *Fix risk shares $\{\delta_t^i\}_{i\in\mathcal{N}_t}$ satisfying $\sum_i \delta_t^i \le \delta_t$, and define $\bar{d}_t^i(a)$ by Eq. 5. For any allocation determined by $\mathcal{H}_t$, condition Eq. 6 implies the latency chance constraint Eq. 2c.*

Proof. Fix the information $\mathcal{H}_t$ and let $\mathbf{a}_t = (a_t^i)_{i\in\mathcal{N}_t}$ be the selected allocation. Since the allocation is determined by $\mathcal{H}_t$, conditioning additionally on $\mathbf{a}_t$ does not change the conditional probabilities below.

For each node $i \in \mathcal{N}_t$, define the event

$$E_t^i := \left\{ D_t^i(a_t^i) > \bar{d}_t^i(a_t^i) \right\},$$

which occurs when its execution latency exceeds its assigned quantile. By Eq. 5,

$$\Pr(E_t^i \mid \mathcal{H}_t) \le \delta_t^i.$$

The event that at least one node exceeds its assigned quantile is $\bigcup_{i\in\mathcal{N}_t} E_t^i$. Applying the conditional union bound,

$$\Pr\left(\bigcup_{i\in\mathcal{N}_t} E_t^i \,\middle|\, \mathcal{H}_t \right) \le \sum_{i\in\mathcal{N}_t} \Pr(E_t^i \mid \mathcal{H}_t) \le \sum_{i\in\mathcal{N}_t} \delta_t^i \le \delta_t.$$

Consequently, the complementary event

$$\mathcal{E}_t := \bigcap_{i\in\mathcal{N}_t} (E_t^i)^c = \left\{ D_t^i(a_t^i) \le \bar{d}_t^i(a_t^i) \text{ for every } i \in \mathcal{N}_t \right\}$$

satisfies $\Pr(\mathcal{E}_t \mid \mathcal{H}_t) \ge 1 - \delta_t$.

On $\mathcal{E}_t$, every source-to-sink path $P \in \mathcal{P}(G_t)$ satisfies

$$\sum_{i\in P} D_t^i(a_t^i) \le \sum_{i\in P} \bar{d}_t^i(a_t^i).$$

Taking the maximum over all such paths and using Eq. 6 yields

$$L_t = \max_{P\in\mathcal{P}(G_t)} \sum_{i\in P} D_t^i(a_t^i) \le \max_{P\in\mathcal{P}(G_t)} \sum_{i\in P} \bar{d}_t^i(a_t^i) \le \mu_t - d_t^{\text{plan}}.$$

Thus $\mathcal{E}_t \subseteq \{L_t \le \mu_t - d_t^{\text{plan}}\}$, and therefore

$$\Pr\left(L_t \le \mu_t - d_t^{\text{plan}} \,\middle|\, \mathcal{H}_t, \mathbf{a}_t \right) \ge \Pr(\mathcal{E}_t \mid \mathcal{H}_t) \ge 1 - \delta_t.$$

This establishes Eq. 2c. The union bound holds for arbitrarily dependent events, so no independence assumption among node latencies is required. □

A.2.2 Average-budget guarantee

Assumption A.2 (Oracle environment and conservative feasibility)**.** Let ξ_t denote the current query context, including the query, task graph, candidate executor sets, resource requirements, planning overheads, and node risk shares. The contexts $\{\xi_t\}$ are independent and identically distributed. The node risk shares are determined by a fixed rule from the current context.

Conditional on the current context and selected allocation, the joint distribution of execution latencies and API costs is time-invariant and does not depend on past rounds. For every admissible allocation, $1 \le |\mathcal{N}_t| \le n_{\max}$ and $0 \le C_t(\mathbf{a}_t, \mathbf{h}_t) \le C_{\max}$ almost surely.

There exists a stationary policy π^0, depending only on the current context and, if randomized, fresh independent randomness, whose induced assignment and finish-time variables satisfy

$$(\mathbf{z}_t^0, \mathbf{F}_t^0) \in \mathcal{F}_t(\mathbf{d}_t) \qquad \text{almost surely,}$$

where $\mathbf{d}_t$ contains the true node-level latency quantiles under the same risk-allocation rule used by the oracle MILP. Moreover,

$$\mathbb{E}_{\pi^0}\left[C_t(\mathbf{a}_t^0, \mathbf{h}_t^0)\right] \leq \gamma,$$

where total cost includes planning, execution, and verification.

This assumption requires budget feasibility within the conservative feasible set used by the oracle optimizer, because for JOVE, the node-quantile reduction may exclude some chance-feasible allocations. The assumption does not require independence among node latencies within a query.

Proposition A.3 (Long-term budget feasibility)**.** *Under Assumption A.2, suppose each query maximizes Eq. 4 over $\mathcal{F}_t(\mathbf{d}_t)$ using true expected costs and qualities. Let the multiplier update be*

$$\lambda_{t+1} = \left[\lambda_t + \alpha_\lambda(c_t - \gamma)\right]_+, \qquad \lambda_1 = 0,$$

where c_t is the realized cost of solving query x_t, and $\alpha_\lambda > 0$ controls the tradeoff between performance and cost. Then $\mathbb{E}[\lambda_{T+1}]/T \to 0$, and the long-term budget constraint Eq. 2b holds:

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[C_t(\mathbf{a}_t, \mathbf{h}_t)] \leq \gamma.$$

Proof. Let $\mathcal{H}_{t-1}$ denote the history before the current context arrives, so λ_t is $\mathcal{H}_{t-1}$-measurable. Define the expected average node quality as

$$U_t(\mathbf{a}_t) = \frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t} \bar{y}_t^i(a_t^i), \qquad 0 \leq U_t(\mathbf{a}_t) \leq 1.$$

Let $(\mathbf{a}_t^0, \mathbf{h}_t^0)$ denote the allocation selected by a feasible policy π^0, and $(\mathbf{a}_t^*, \mathbf{h}_t^*)$ denote the optimal allocation to maximize the priced objective Eq. 4.

Since current-query information is identically distributed across rounds and the conditional cost laws do not change, the expected cost of policy π_0 is a constant:

$$\mathbb{E}[C_t(\mathbf{a}_t^0, \mathbf{h}_t^0)] = m_0.$$

Its long-term budget feasibility therefore implies

$$m_0 = \limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[C_t(\mathbf{a}_t^0, \mathbf{h}_t^0)] \leq \gamma.$$

Moreover, λ_t depends only on past observations, whereas the conditional expected cost under π^0 depends only on the independent current-query information. Hence

$$\mathbb{E}[\lambda_t C_t(\mathbf{a}_t^0, \mathbf{h}_t^0)] = m_0\,\mathbb{E}[\lambda_t].$$

The planner cost is common to all allocations of the current query, so including it does not change the oracle optimizer. Oracle optimality, followed by taking expectations, gives

$$\mathbb{E}[U_t(\mathbf{a}_t^*) - \lambda_t c_t] \geq \mathbb{E}[U_t(\mathbf{a}_t^0) - \lambda_t C_t(\mathbf{a}_t^0, \mathbf{h}_t^0)].$$

Rearranging and using $0 \leq U_t \leq 1$, we obtain

$$\begin{aligned}\mathbb{E}[\lambda_t(c_t - \gamma)] &\leq 1 + \mathbb{E}[\lambda_t C_t(\mathbf{a}_t^0, \mathbf{h}_t^0)] - \gamma\mathbb{E}[\lambda_t] \\ &= 1 + (m_0 - \gamma)\mathbb{E}[\lambda_t] \leq 1.\end{aligned}$$

Set

$$B = \frac{\alpha_\lambda}{2}\max\{\gamma^2, (C_{\max} - \gamma)^2\}.$$

Using $[u]_+^2 \leq u^2$ and $0 \leq c_t \leq C_{\max}$, the multiplier update yields

$$\frac{\lambda_{t+1}^2 - \lambda_t^2}{2\alpha_\lambda} \leq \lambda_t(c_t - \gamma) + B.$$

Taking expectations, summing over $t = 1, \ldots, T$, and using $\lambda_1 = 0$ gives

$$\mathbb{E}[\lambda_{T+1}^2] \leq 2\alpha_\lambda (B+1)T.$$

Consequently,

$$\frac{\mathbb{E}[\lambda_{T+1}]}{T} \leq \frac{\sqrt{\mathbb{E}[\lambda_{T+1}^2]}}{T} \leq \sqrt{\frac{2\alpha_\lambda(B+1)}{T}} \longrightarrow 0.$$

Finally, the update also implies

$$\lambda_{t+1} - \lambda_t \geq \alpha_\lambda (c_t - \gamma).$$

Summing over rounds and taking expectations establishes

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[C_t(\mathbf{a}_t^*, \mathbf{h}_t^*)] = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[c_t] \leq \gamma + \frac{\mathbb{E}[\lambda_{T+1}]}{\alpha_\lambda T}.$$

Taking the upper limit as $T \to \infty$ proves the long-term budget constraint. □

A.2.3 LLM QUALITY LEARNING ANALYSIS

Selection of β_t. To facilitate the theoretical analysis, we define a standard confidence radius for regularized linear regression (Abbasi-Yadkori et al., 2011) to ensure that the uncertainty bonus $\beta_t u_t^i(a)$ covers the quality-estimation error with high probability:

$$\beta_t = \sqrt{\lambda_{\text{reg}}} S + R\sqrt{2\log(1/\delta_{\text{conf}}) + d\log\left(1 + \frac{|\mathcal{D}_{t-1}|L^2}{d\lambda_{\text{reg}}}\right)}. \tag{15}$$

The first term accounts for regularization bias, while the second accounts for noise in the verified feedback. Under Assumption 3.3, this β_t guarantees the following inequality with probability at least $1 - \delta_{\text{conf}}$:

$$\|\hat{\theta}_t - \theta^\star\|_{G_{t-1}} \leq \beta_t \qquad \text{for all } t,$$

where $\|v\|_G = \sqrt{v^\top G v}$. This can be proved following the same steps as in (Abbasi-Yadkori et al., 2011, Theorem 2). The result applies to our verified observations because their features are selected before observing the corresponding conditionally zero-mean, sub-Gaussian label errors ϵ_t^i as defined in Assumption 3.3. Consequently, by Cauchy–Schwarz, we can show:

$$\left|\phi_t^i(a)^\top \hat{\theta}_t - \bar{y}_t^i(a)\right| \leq \beta_t u_t^i(a).$$

β_t calibrates the uncertainty bonus so that the resulting UCB is an upper bound on true model quality on this high-probability event. For a fixed experimental horizon T, we may use a constant $\beta_t = \beta$ for all $t \leq T$. The theoretical guarantees remain valid provided that β upper-bounds the confidence radii in Eq. 15 over this horizon. Choosing a larger constant β preserves the confidence guarantee, although it may encourage excessive exploration and increase the regret bound. As verified observations accumulate, uncertainty decreases in the feature directions they cover, reducing the corresponding exploration bonuses.

Sublinear regret in quality learning. We now prove Theorem 3.5. We first bound the cumulative uncertainty of the verified task-model pairs. Notice that for each query, its selected nodes for verification are decided all at once by solving the MILP as described in JOVE. Let $\mathcal{V}_t = \{i \in \mathcal{N}_t : h_t^i = 1\}$ denote the set of nodes selected for verification, and write $\phi_{t,i} = \phi_t^i(a_t^i)$ for convenience. From round $t-1$ to t, the information matrix updates as

$$G_t = G_{t-1} + \sum_{i \in \mathcal{V}_t} \phi_{t,i}\phi_{t,i}^\top, \qquad G_0 = \lambda_{\text{reg}} I.$$

Recall that

$$[u_t^i(a_t^i)]^2 = \phi_{t,i}^\top G_{t-1}^{-1} \phi_{t,i}.$$

Define

$$N := Tn_{\max}, \qquad C_{\text{bat}} := \left(1 + \frac{(n_{\max}-1)L^2}{\lambda_{\text{reg}}}\right)\left(1 + \frac{L^2}{\lambda_{\text{reg}}}\right),$$

and

$$\Gamma_T := C_{\mathrm{bat}}\, d \log\left(1 + \frac{NL^2}{d\lambda_{\mathrm{reg}}}\right).$$

Here, N upper-bounds the total number of nodes over T queries, and C_{bat} accounts for the use of a shared information matrix within each query.

Lemma A.4 (Batched uncertainty bound)**.** *Under Assumptions 3.3–3.4,*

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i [u_t^i(a_t^i)]^2 \le \Gamma_T, \qquad \sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i I_t^i(a_t^i) \le \frac{\Gamma_T}{2}.$$

Proof. Step 1: Compare batch and sequential uncertainties. For each query t, order its verified features as $\phi_{t,1}, \dots, \phi_{t,b_t}$, where $b_t = |\mathcal{V}_t| \le n_{\max}$. Define the intermediate matrices

$$V_{t,0} = G_{t-1}, \qquad V_{t,j} = V_{t,j-1} + \phi_{t,j}\phi_{t,j}^\top, \quad j = 1, \dots, b_t.$$

Thus, $V_{t,b_t} = G_t$. These matrices are introduced only for the analysis; the algorithm computes every uncertainty in query t using G_{t-1}.

Since $\|\phi_{t,k}\|_2 \le L$, we have $\phi_{t,k}\phi_{t,k}^\top \preceq L^2 I$. Also, $G_{t-1} \succeq \lambda_{\mathrm{reg}} I$ as it only adds positive semi-definite matrices per round, so $I \preceq G_{t-1}/\lambda_{\mathrm{reg}}$. Consequently, for each $j = 1, \dots, b_t$,

$$\begin{aligned} V_{t,j-1} &= G_{t-1} + \sum_{k=1}^{j-1} \phi_{t,k}\phi_{t,k}^\top \\ &\preceq G_{t-1} + (j-1)L^2 I \\ &\preceq \left(1 + \frac{(j-1)L^2}{\lambda_{\mathrm{reg}}}\right) G_{t-1} \\ &\preceq \underbrace{\left(1 + \frac{(n_{\max}-1)L^2}{\lambda_{\mathrm{reg}}}\right)}_{=:\,\alpha_{\mathrm{bat}}} G_{t-1}. \end{aligned}$$

Inversion reverses the positive-definite matrix order. Therefore,

$$V_{t,j-1}^{-1} \succeq \alpha_{\mathrm{bat}}^{-1} G_{t-1}^{-1}, \qquad \text{or equivalently} \qquad G_{t-1}^{-1} \preceq \alpha_{\mathrm{bat}} V_{t,j-1}^{-1}.$$

Writing

$$q_{t,j} := \phi_{t,j}^\top V_{t,j-1}^{-1} \phi_{t,j},$$

By using the above inequality, we obtain

$$\phi_{t,j}^\top G_{t-1}^{-1} \phi_{t,j} \le \alpha_{\mathrm{bat}} q_{t,j}.$$

Summing over all verified nodes gives

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i [u_t^i(a_t^i)]^2 \le \alpha_{\mathrm{bat}} \sum_{t=1}^{T}\sum_{j=1}^{b_t} q_{t,j}.$$

Step 2: Bound sequential uncertainties by log-determinant increments. Because $V_{t,j-1} \succeq \lambda_{\mathrm{reg}} I$,

$$0 \le q_{t,j} \le \frac{\|\phi_{t,j}\|_2^2}{\lambda_{\mathrm{reg}}} \le \frac{L^2}{\lambda_{\mathrm{reg}}} =: r.$$

Notice that for any $x \in [0, r]$,

$$\log(1+x) = \int_0^x \frac{1}{1+s}\, ds \ge \frac{x}{1+r}.$$

Hence $q_{t,j} \le (1+r)\log(1+q_{t,j})$. The matrix determinant lemma gives

$$\begin{aligned} \det(V_{t,j}) &= \det(V_{t,j-1})\left(1 + \phi_{t,j}^\top V_{t,j-1}^{-1}\phi_{t,j}\right) \\ &= \det(V_{t,j-1})(1 + q_{t,j}), \end{aligned}$$

and therefore

$$\log(1+q_{t,j}) = \log\det(V_{t,j}) - \log\det(V_{t,j-1}).$$

Summing these increments first within each query and then across queries yields

$$\begin{aligned}\sum_{t=1}^{T}\sum_{j=1}^{b_t}\log(1+q_{t,j}) &= \sum_{t=1}^{T}[\log\det(V_{t,b_t}) - \log\det(V_{t,0})] \\ &= \sum_{t=1}^{T}[\log\det(G_t) - \log\det(G_{t-1})] \\ &= \log\frac{\det(G_T)}{\det(G_0)}.\end{aligned}$$

Queries with $b_t = 0$ contribute zero to both sides. Combining this identity with Step 1, and using $C_{\mathrm{bat}} = \alpha_{\mathrm{bat}}(1+r)$, gives

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i[u_t^i(a_t^i)]^2 \le C_{\mathrm{bat}}\log\frac{\det(G_T)}{\det(G_0)}.$$

Step 3: Bound the determinant ratio. The total number of verified observations satisfies $\sum_{t=1}^{T} b_t \le Tn_{\max} = N$. Using $\mathrm{tr}(\phi\phi^\top) = \|\phi\|_2^2$, we obtain

$$\begin{aligned}\mathrm{tr}(G_T) &= d\lambda_{\mathrm{reg}} + \sum_{t=1}^{T}\sum_{j=1}^{b_t}\|\phi_{t,j}\|_2^2 \\ &\le d\lambda_{\mathrm{reg}} + NL^2.\end{aligned}$$

Let $\mu_1,\dots,\mu_d > 0$ be the eigenvalues of G_T. The arithmetic–geometric mean inequality implies

$$\det(G_T) = \prod_{\ell=1}^{d}\mu_\ell \le \left(\frac{\sum_{\ell=1}^{d}\mu_\ell}{d}\right)^d = \left(\frac{\mathrm{tr}(G_T)}{d}\right)^d \le \left(\lambda_{\mathrm{reg}} + \frac{NL^2}{d}\right)^d.$$

Since $\det(G_0) = \lambda_{\mathrm{reg}}^d$,

$$\log\frac{\det(G_T)}{\det(G_0)} \le d\log\left(1 + \frac{NL^2}{d\lambda_{\mathrm{reg}}}\right).$$

Substituting this into the bound from Step 2 proves

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i[u_t^i(a_t^i)]^2 \le \Gamma_T.$$

Step 4: Bound the cumulative information gain. By the definition of $I_t^i(a)$ and the inequality $\log(1+x) \le x$ for $x \ge 0$,

$$I_t^i(a) = \frac{1}{2}\log\left(1 + [u_t^i(a)]^2\right) \le \frac{1}{2}[u_t^i(a)]^2.$$

Multiplying by $h_t^i \ge 0$ and summing gives

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i I_t^i(a_t^i) \le \frac{1}{2}\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i[u_t^i(a_t^i)]^2 \le \frac{\Gamma_T}{2},$$

which proves the second claim. □

Here we provide a lemma to convert the lower bound of $N = Tn_{\max}$ to the lower bound of the horizon T.

Lemma A.5 (Sufficient horizon length). *Let $N = Tn_{\max}$ and recall that*

$$C_{\mathrm{bat}} = \left(1 + \frac{(n_{\max}-1)L^2}{\lambda_{\mathrm{reg}}}\right)\left(1 + \frac{L^2}{\lambda_{\mathrm{reg}}}\right), \qquad \Gamma_T = C_{\mathrm{bat}} d \log\left(1 + \frac{NL^2}{d\lambda_{\mathrm{reg}}}\right).$$

Under $\lambda_{\mathrm{reg}} \geq L^2$, we have $1 \leq C_{\mathrm{bat}} \leq 2n_{\max}$. Furthermore, $\Gamma_T \leq N$ whenever

$$N \geq 4C_{\mathrm{bat}} d \log(4C_{\mathrm{bat}}). \tag{16}$$

In particular, a sufficient condition is

$$T \geq 8d \log(8n_{\max}).$$

Proof. Step 1: Bound the batching constant. Write

$$r := \frac{L^2}{\lambda_{\mathrm{reg}}} \in [0,1], \qquad C := C_{\mathrm{bat}}.$$

Since $n_{\max} \geq 1$,

$$1 \leq 1 + (n_{\max}-1)r \leq n_{\max}, \qquad 1 \leq 1 + r \leq 2.$$

Multiplying these inequalities yields

$$1 \leq C = (1 + (n_{\max}-1)r)(1+r) \leq 2n_{\max}.$$

Step 2: Establish $\Gamma_T \leq N$. Set $x := N/d$. Then

$$\Gamma_T = Cd \log(1 + rx), \qquad N = dx.$$

Because $r \leq 1$ and the logarithm is increasing, it suffices to show that $C \log(1+x) \leq x$.

By concavity, $\log y$ lies below its tangent at $y = 2C$. Thus, for every $y > 0$,

$$\log y \leq \log(2C) + \frac{y - 2C}{2C}.$$

Taking $y = 1 + x$ and using $C \geq 1$ gives

$$\begin{aligned}
\log(1+x) &\leq \log(2C) + \frac{1 + x - 2C}{2C} \\
&= \log(2C) + \frac{x}{2C} + \frac{1 - 2C}{2C} \\
&\leq \log(2C) + \frac{x}{2C}.
\end{aligned}$$

Condition Eq. 16 is equivalent to $x \geq 4C \log(4C)$. Under this condition,

$$\frac{x}{2C} \geq 2 \log(4C) \geq \log(2C),$$

where the last inequality holds because $C \geq 1$. Consequently,

$$\log(1+x) \leq \frac{x}{2C} + \frac{x}{2C} = \frac{x}{C}.$$

Substituting into the expression for Γ_T yields

$$\Gamma_T = Cd \log(1 + rx) \leq Cd \log(1+x) \leq Cd \frac{x}{C} = dx = N.$$

Step 3: Derive a condition directly on T. The function $f(c) = c \log(4c)$ is increasing for $c \geq 1$, since

$$f'(c) = \log(4c) + 1 > 0.$$

Using $1 \leq C \leq 2n_{\max}$, we therefore obtain

$$4Cd \log(4C) \leq 4(2n_{\max}) d \log(8n_{\max}) = 8n_{\max} d \log(8n_{\max}).$$

If $T \geq 8d \log(8n_{\max})$, then

$$N = n_{\max} T \geq 8n_{\max} d \log(8n_{\max}) \geq 4Cd \log(4C).$$

Thus, condition Eq. 16 holds, and Step 2 implies $\Gamma_T \leq N$. □

Proof of Theorem 3.5

Proof. Write $n_t := |\mathcal{N}_t|$ and $N := Tn_{\max}$. Then $\sum_{t=1}^{T} n_t \le N$. By the theorem statement, $c_v > 0$. Define $\beta_T^{\max} := \max_{1\le t\le T} \beta_t$. Use the confidence radii in Eq. 15. Order the verified observations within each query before observing their labels, and concatenate these orders across queries. The selected features and verification decisions are determined before observing the corresponding labels. Under Assumption 3.3, each label error is zero-mean and R-sub-Gaussian given the selected features and all preceding observations, including earlier labels in the same query. Thus, the confidence-ellipsoid result of Abbasi-Yadkori et al. (2011, Theorem 2), together with the determinant–trace bound, gives

$$\mathbb{P}(\mathcal{E}) \ge 1 - \delta_{\text{conf}}, \qquad \mathcal{E} := \left\{ \|\hat{\theta}_t - \theta^\star\|_{G_{t-1}} \le \beta_t \text{ for every } t \le T \right\}.$$

For every candidate task–model pair, Cauchy–Schwarz gives

$$\begin{aligned} \left| \phi_t^i(a)^\top \hat{\theta}_t - \bar{y}_t^i(a) \right| &= \left| \phi_t^i(a)^\top (\hat{\theta}_t - \theta^\star) \right| \\ &\le \|\phi_t^i(a)\|_{G_{t-1}^{-1}} \|\hat{\theta}_t - \theta^\star\|_{G_{t-1}} \\ &\le \beta_t u_t^i(a). \end{aligned}$$

Consequently, the untruncated optimistic estimate

$$s_t^i(a) := \phi_t^i(a)^\top \hat{\theta}_t + \beta_t u_t^i(a)$$

satisfies

$$\bar{y}_t^i(a) \le s_t^i(a) \le \bar{y}_t^i(a) + 2\beta_t u_t^i(a).$$

Since $\bar{y}_t^i(a) \in [0,1]$, clipping $s_t^i(a)$ to $[0,1]$ preserves its lower bound by $\bar{y}_t^i(a)$ and cannot increase its excess over $\bar{y}_t^i(a)$. Hence

$$0 \le \text{UCB}_t^i(a) - \bar{y}_t^i(a) \le 2\beta_t u_t^i(a) \tag{17}$$

simultaneously for all rounds and candidate pairs.

The assumed horizon condition and Lemma A.5 imply $\Gamma_T \le N$. For $\Gamma_T > 0$, define

$$\tau_T := \left(\frac{\Gamma_T}{N} \right)^{1/3} \in (0,1], \qquad K_T := \frac{2n_{\max} c_v}{\log(1+\tau_T^2)}.$$

The choice of τ_T gives the identity $\Gamma_T = N\tau_T^3$, which will be used to simplify the cumulative regret bound. Consider any selected task–model pair satisfying $u_t^i(a_t^i) > \tau_T$. By the definition of the information gain,

$$I_t^i(a_t^i) = \frac{1}{2} \log\left(1 + [u_t^i(a_t^i)]^2\right) > \frac{1}{2} \log(1+\tau_T^2).$$

Assumption 3.4 also gives

$$0 \le \lambda_t \hat{\widetilde{m}}_t^{\,i} \le \lambda_{\max} \widetilde{m}_{\max} = c_v.$$

Since $k_v \ge K_T > 0$, we obtain

$$\begin{aligned} \frac{k_v}{n_t} I_t^i(a_t^i) - \lambda_t \hat{\widetilde{m}}_t^{\,i} &\ge \frac{K_T}{n_t} I_t^i(a_t^i) - c_v \\ &\ge \frac{K_T}{n_{\max}} I_t^i(a_t^i) - c_v \\ &> \frac{K_T}{2n_{\max}} \log(1+\tau_T^2) - c_v \\ &= 0. \end{aligned}$$

For a fixed executor choice, changing h_t^i from 0 to 1 changes the online objective by precisely this quantity, because the verification bonus in Eq. 11 is normalized by n_t, while the priced verification

cost is not. This change leaves feasibility unchanged. Therefore, if $h_t^i = 0$, this change would strictly improve the objective, contradicting exact optimality. We conclude that

$$u_t^i(a_t^i) > \tau_T \quad \Longrightarrow \quad h_t^i = 1. \tag{18}$$

For a feasible allocation $\mathbf{z}$, define

$$\widehat{J}_t(\mathbf{z}) := \sum_{i\in\mathcal{N}_t} \sum_{a\in\mathcal{A}_t^i} \sum_{\eta\in\{0,1\}} \left[\frac{1}{n_t}\mathrm{UCB}_t^i(a) - \lambda_t \hat{c}_t^i(a,\eta)\right] z^i(a,\eta),$$

and

$$B_t(\mathbf{z}) := \frac{1}{n_t} \sum_{i\in\mathcal{N}_t} \sum_{a\in\mathcal{A}_t^i} \sum_{\eta\in\{0,1\}} \eta I_t^i(a) z^i(a,\eta).$$

By Eq. 11, JOVE maximizes $\widehat{J}_t(\mathbf{z}) + k_v B_t(\mathbf{z})$ over the same feasible set used by the comparator.

On $\mathcal{E}$, optimism and the nonnegativity of the allocation variables imply

$$J_t(\mathbf{z}_t^\star) \le \widehat{J}_t(\mathbf{z}_t^\star).$$

Exact optimality of JOVE gives

$$\widehat{J}_t(\mathbf{z}_t) + k_v B_t(\mathbf{z}_t) \ge \widehat{J}_t(\mathbf{z}_t^\star) + k_v B_t(\mathbf{z}_t^\star).$$

Rearranging this inequality and using optimism yields

$$\begin{aligned} J_t(\mathbf{z}_t^\star) - J_t(\mathbf{z}_t) &\le \widehat{J}_t(\mathbf{z}_t^\star) - J_t(\mathbf{z}_t) \\ &\le \widehat{J}_t(\mathbf{z}_t) - J_t(\mathbf{z}_t) + k_v \left[B_t(\mathbf{z}_t) - B_t(\mathbf{z}_t^\star)\right] \\ &\le \widehat{J}_t(\mathbf{z}_t) - J_t(\mathbf{z}_t) + k_v B_t(\mathbf{z}_t), \end{aligned}$$

where the last step uses $k_v \ge 0$ and $B_t(\mathbf{z}_t^\star) \ge 0$.

At the same allocation $\mathbf{z}_t$, the resource-cost terms in $\widehat{J}_t$ and J_t are identical and cancel. Since exactly one executor–verification action is selected for each node,

$$\widehat{J}_t(\mathbf{z}_t) - J_t(\mathbf{z}_t) = \frac{1}{n_t} \sum_{i\in\mathcal{N}_t} \left[\mathrm{UCB}_t^i(a_t^i) - \bar{y}_t^i(a_t^i)\right] \le \frac{2\beta_t}{n_t} \sum_{i\in\mathcal{N}_t} u_t^i(a_t^i),$$

and

$$B_t(\mathbf{z}_t) = \frac{1}{n_t} \sum_{i\in\mathcal{N}_t} h_t^i I_t^i(a_t^i).$$

Combining these expressions proves

$$J_t(\mathbf{z}_t^\star) - J_t(\mathbf{z}_t) \le \frac{2\beta_t}{n_t} \sum_{i\in\mathcal{N}_t} u_t^i(a_t^i) + \frac{k_v}{n_t} \sum_{i\in\mathcal{N}_t} h_t^i I_t^i(a_t^i). \tag{19}$$

Only the resource terms evaluated at the same allocation cancel; no equality of costs between JOVE and the comparator is required.

Partition the executed nodes according to whether their uncertainty exceeds τ_T:

$$\mathcal{N}_t^{>} := \{i \in \mathcal{N}_t : u_t^i(a_t^i) > \tau_T\}, \qquad \mathcal{N}_t^{\le} := \{i \in \mathcal{N}_t : u_t^i(a_t^i) \le \tau_T\}.$$

For the nodes in $\mathcal{N}_t^{\le}$, each uncertainty is at most τ_T. Since there are at most N executed nodes in total,

$$\sum_{t=1}^{T} \sum_{i\in\mathcal{N}_t^{\le}} u_t^i(a_t^i) \le \tau_T \sum_{t=1}^{T} |\mathcal{N}_t^{\le}| \le N\tau_T.$$

For every node in $\mathcal{N}_t^{>}$, Eq. 18 guarantees $h_t^i = 1$. Moreover, $u_t^i(a_t^i) > \tau_T$ implies

$$u_t^i(a_t^i) \le \frac{[u_t^i(a_t^i)]^2}{\tau_T}.$$

Therefore, Lemma A.4 gives

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t^>} u_t^i(a_t^i) \le \frac{1}{\tau_T}\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t^>}[u_t^i(a_t^i)]^2 = \frac{1}{\tau_T}\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t^>} h_t^i[u_t^i(a_t^i)]^2$$
$$\le \frac{1}{\tau_T}\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i[u_t^i(a_t^i)]^2$$
$$\le \frac{\Gamma_T}{\tau_T}.$$

Adding the two contributions yields

$$\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} u_t^i(a_t^i) \le N\tau_T + \frac{\Gamma_T}{\tau_T}. \tag{20}$$

Sum Eq. 19 over t. Using $n_t \ge 1$, $\beta_t \le \beta_T^{\max}$, and the nonnegativity of uncertainty and information gain, we obtain

$$\mathcal{R}_T \le 2\beta_T^{\max}\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} u_t^i(a_t^i) + k_v\sum_{t=1}^{T}\sum_{i\in\mathcal{N}_t} h_t^i I_t^i(a_t^i) \le 2\beta_T^{\max}\left(N\tau_T + \frac{\Gamma_T}{\tau_T}\right) + \frac{k_v\Gamma_T}{2},$$

where the last inequality uses Eq. 20 and the second bound in Lemma A.4.

We now bound the two terms separately. Because $\Gamma_T = N\tau_T^3$ and $\tau_T \le 1$,

$$N\tau_T + \frac{\Gamma_T}{\tau_T} = N\tau_T + N\tau_T^2 \le 2N\tau_T.$$

Hence

$$2\beta_T^{\max}\left(N\tau_T + \frac{\Gamma_T}{\tau_T}\right) \le 4\beta_T^{\max}N\tau_T.$$

For the information-gain term, note that $0 \le \tau_T^2 \le 1$ implies

$$\log(1+\tau_T^2) = \int_0^{\tau_T^2}\frac{1}{1+s}\,ds \ge \frac{\tau_T^2}{2}.$$

Using the upper endpoint $k_v \le 2K_T$ gives

$$\frac{k_v\Gamma_T}{2} \le K_T\Gamma_T = \frac{2n_{\max}c_v\Gamma_T}{\log(1+\tau_T^2)} \le \frac{4n_{\max}c_v\Gamma_T}{\tau_T^2} = 4n_{\max}c_vN\tau_T.$$

Combining the two bounds and substituting the definition of τ_T, we conclude that

$$\mathcal{R}_T \le 4(n_{\max}c_v + \beta_T^{\max})N\tau_T$$
$$= 4(n_{\max}c_v + \beta_T^{\max})N^{2/3}\Gamma_T^{1/3}$$
$$= 4\left(n_{\max}c_v + \max_{t\le T}\beta_t\right)(n_{\max}T)^{2/3}\Gamma_T^{1/3}.$$

This inequality holds on $\mathcal{E}$, whose probability is at least $1-\delta_{\mathrm{conf}}$.

Since $|\mathcal{D}_{t-1}| \le N$, the confidence radii satisfy the deterministic upper bound

$$\beta_T^{\max} \le \sqrt{\lambda_{\mathrm{reg}}}S + R\sqrt{2\log(1/\delta_{\mathrm{conf}}) + d\log\left(1+\frac{NL^2}{d\lambda_{\mathrm{reg}}}\right)}.$$

For fixed $n_{\max}$, L, S, R, λ_{reg}, $\lambda_{\max}$, $\widetilde{m}_{\max}$, and δ_{conf}, we have

$$\Gamma_T = O(d\log(1+T)), \qquad n_{\max}c_v + \beta_T^{\max} = O\left(\sqrt{d\log(1+T)}\right).$$

Substituting these estimates into the regret bound gives

$$\begin{aligned}\mathcal{R}_T &= O\Big([d\log(1+T)]^{1/2}T^{2/3}[d\log(1+T)]^{1/3}\Big)\\ &= O\Big(d^{5/6}T^{2/3}[\log(1+T)]^{5/6}\Big).\end{aligned}$$

In particular, for fixed d,

$$\frac{\mathcal{R}_T}{T} = O\Big(T^{-1/3}[\log(1+T)]^{5/6}\Big) \longrightarrow 0.$$

□

The lower endpoint ensures that sufficiently uncertain selections are verified, while the upper endpoint controls the cumulative information-gain bonus. The factor two is not essential: for any fixed $\rho \geq 1$, choosing $k_v \in [K_T, \rho K_T]$ gives $\mathcal{R}_T \leq (4\beta_T^{\max} + 2\rho n_{\max} c_v)\, N^{2/3}\Gamma_T^{1/3}$ by the same argument. The horizon condition $T \geq 8d\log(8n_{\max})$ is sufficient but need not be necessary: it is used to ensure $\Gamma_T \leq N$ in the proof steps. This condition implies that lower d (the dimension of the feature vector) yields a smaller condition on T, potentially improving the learning speed. Also, we point out that a small d may restrict the expressivity of the feature vector, which may potentially degrade the model quality estimation.

Theory–practice scope. Theorem 3.5 is intended as an existence result showing that a nonempty range of verification weights (k_v) yields sublinear quality-learning regret under the stated assumptions, rather than as a practical tuning rule. Its horizon condition and weight interval are sufficient worst-case conditions arising from conservative concentration and batching bounds. For our 384-dimensional features and $n_{\max} = 5$, the sufficient horizon can exceed the lengths of some experimental sequences. In practice, substantially less exploration may be sufficient. The experiments evaluate JOVE outside the theorem's conservative sufficient regime, while the theorem establishes the existence of verification weights that guarantee sublinear regret under the formal model.

A.3 Implementation details

A.3.1 JOVE procedure

Algorithm 1 JOVE

Require: Planner, fixed verifier, budget γ, step size α_λ, regularization λ_{reg}, confidence radii β_t, and verification weight k_v.

1: Initialize $G_0 = \lambda_{\text{reg}} I$, $\hat{\theta}_1 = \mathbf{0}$, $\lambda_1 = 0$, and resource estimates.
2: **for** each query x_t **do**
3: Generate DAG $\mathcal{G}_t$; record planning cost and reserve allocation time.
4: Use embedding model to form context features $\phi_t^i(a)$ for each node-model pair.
5: Compute $\text{UCB}_t^i(a)$, $I_t^i(a)$, $\hat{c}_t^i(a,\eta)$, and $\hat{d}_t^i(a)$ using prior observations.
6: Solve MILP (Eq. 11) to decide $\mathbf{z}_t$ variables, yielding (a_t^i, h_t^i) for every node.
7: Execute the DAG following dependencies, launching selected verifications as node outputs become available.
8: Deliver the final response without waiting for verification.
9: Collect verification labels and update the quality model via Eq. 8.
10: Update resource histories using observed execution costs, latencies, and verification costs.
11: Compute total realized cost $c_t = m_t^{\text{plan}} + \sum_{i\in\mathcal{N}_t} m_t^i(a_t^i) + \sum_{i:h_t^i=1} \widetilde{m}_t^i$.
12: Update $\lambda_{t+1} = [\lambda_t + \alpha_\lambda(c_t - \gamma)]_+$.
13: **end for**

A.3.2 PLANNER AND VERIFICATION PROMPTS

Planner Prompt

You are a planner for benchmark questions. Decompose the user prompt into {node_count} small executable tasks arranged as a DAG. Return JSON only; no markdown or extra text. Do not solve the question, compute the final answer, or include reasoning. For multi−hop factual QA, split anchor/entity lookup from the final attribute lookup. For math, science, multiple−choice, or reasoning questions, split solution reasoning from final answer extraction. Each task needs lowercase fields "id", "description", "predecessors", "type", "output_format", "input_template". "type" must be one of: {task_groups_list}. Do not create task nodes whose purpose is to verify, check, validate, confirm, or audit another task output. Use precise output_format values such as "full name only", "date only", "year only", "number only", "integer only", "option letter only", or "short phrase only". The sink task output_format must satisfy any final−format instruction in the user message. The sink input_template must ask for the original final answer using predecessors. Keep dependencies minimal and acyclic.
Schema: {"tasks":[{"id":"t1","description":"...","predecessors":[],"type":"Anchor Identification","output_format":"full name only","input_template":"..."}]}.
Question: {...}

Verifier Prompt

Original user prompt: {prompt}. Task to verify ({task.id}): {instruction}. Output format requirement: {task.output_format}. Executor output: {executor_output}.
Is the executor output factually correct for this task?
decision=true only if the required result is right given the original prompt and dependencies.
decision=false if a name/date/number/place/option is wrong or invented, the answer is a fluent guess you cannot verify, the output is empty/refusal/off−task, or only the format is right.
If you are unsure, prefer decision=false.
Return JSON only: {"reason": "short reason", "decision": true/false}.

A.3.3 EMBEDDING MODEL

We use a frozen text embedding as the feature map for the LinUCB head. Specifically, to construct $\phi_t^i(a)$, we encode the node's description, descriptions of required node outputs, and LLM API description with Sentence-Transformers all-MiniLM-L6-v2 (Reimers & Gurevych, 2019), which produces a 384-dimensional unit-normalized feature vector. We choose this model because it is small ($\sim$22M parameters) and runs fast even on CPU, reducing the reserved time for executor allocation. Larger modern embedding models (e.g., the Qwen3 Embedding family, such as the 0.6B or 4B variants with up to 1024- or 2560-dimensional outputs) may improve retrieval-oriented embedding quality, but they are substantially slower and more memory-intensive to evaluate for every task–model pair, increase the LinUCB feature dimension (and thus the cost of maintaining and inverting the covariance matrix). Moreover, Theorem 3.5 gives a larger sufficient horizon and a looser regret bound as d increases, suggesting potentially slower learning. These considerations motivate our choice of a compact encoder.

A.3.4 RESOURCE ESTIMATORS

JOVE's MILP uses online estimates of expected execution and verification costs, together with executor latency quantiles. For each executor API a, we maintain a bounded history of recent calls,

$$\Upsilon_{t-1}(a) \subseteq \left\{(\phi_s^j(a), m_s^j(a), d_s^j(a)) : s < t,\ j \in \mathcal{N}_s,\ a_s^j = a\right\}.$$

$\Upsilon_{t-1}(a)$ contains the execution records for model a collected before query t. Each record (ϕ, m, d) contains the feature vector, realized API cost, and observed latency. We estimate expected cost and latency quantiles by weighting historical observations according to their feature similarity to the

current task. For a nonempty history, define

$$\hat{m}^i_{t,\mathrm{emp}}(a) = \sum_{(\phi,m,d)\in\Upsilon_{t-1}(a)} w\left(\phi, \phi^i_t(a)\right) m \quad \text{with} \quad w\left(\phi, \phi^i_t(a)\right) = \frac{\exp\left(\phi^i_t(a)^\top \phi\right)}{\sum_{(\phi',m',d')\in\Upsilon_{t-1}(a)} \exp\left(\phi^i_t(a)^\top \phi'\right)}. \tag{21}$$

For unit-normalized vectors, the inner product in Eq. 21 is cosine similarity. Thus, $\hat{m}^i_{t,\mathrm{emp}}(a)$ is a weighted average of the observed costs, with greater weight assigned to calls with features similar to the current task-model pair. Using the same weights, we estimate the latency quantile as

$$\hat{\ell}^i_{t,\mathrm{emp}}(a,\epsilon) = \inf\Big\{u \geq 0 : \sum_{(\phi,m,d)\in\Upsilon_{t-1}(a)} w\big(\phi, \phi^i_t(a)\big)\, \mathbf{1}\{d \leq u\} \geq 1-\epsilon\Big\}. \tag{22}$$

This is the smallest latency threshold containing at least $1-\epsilon$ of the total observation weight.

To handle sparse or empty histories, we initialize each model with a baseline cost $m_0(a)$ and the baseline latency quantile $\ell_0(a, \delta_t/|\mathcal{N}_t|)$. These baselines can be informed by platform statistics or at least one preliminary profiling call for each model.

Let $n_{t-1}(a) = |\Upsilon_{t-1}(a)|$ and define the baseline weight

$$\rho_{t-1}(a) = \frac{1}{1 + n_{t-1}(a)}. \tag{23}$$

For nonempty histories, $\hat{m}^i_{t,\mathrm{emp}}(a)$ and $\hat{\ell}^i_{t,\mathrm{emp}}(a,\epsilon)$ denote the feature-weighted empirical estimates, respectively. The estimates used by JOVE are

$$\hat{m}^i_t(a) = \rho_{t-1}(a) m_0(a) + \big(1-\rho_{t-1}(a)\big)\hat{m}^i_{t,\mathrm{emp}}(a), \tag{24}$$

$$\hat{\ell}^i_t(a,\epsilon) = \rho_{t-1}(a)\ell_0(a,\epsilon) + \big(1-\rho_{t-1}(a)\big)\hat{\ell}^i_{t,\mathrm{emp}}(a,\epsilon), \tag{25}$$

with $\hat{d}^i_t(a) = \hat{\ell}^i_t(a, \delta^i_t)$. When the history is empty, we use the baselines directly. As the retained history grows, the empirical estimates receive greater weight. The verification-cost estimate $\hat{\tilde{m}}^i_t$ is initialized and shrunk analogously using observations from completed verifications.

A.4 Additional experimental Details

For the main experiments in Section 4, we use the executor pool including Llama-3.2-3B-Instruct, Gemma-3-4B-IT, GPT-OSS-20B, Mistral-Small-3.2-24B-Instruct, Qwen3-32B, Llama-3.3-70B-Instruct, GPT-OSS-120B, and Mistral-Large-2512 from OpenRouter platform. The API providers introduce additional randomness of the output quality, latency, and cost into our experiments through fluctuations in KV-cache hit rates, service congestion from concurrent users, network conditions, etc. Network disruptions or other service failures may cause LLM API calls to fail. When a call fails, we treat its output as empty and continue executing the task graph.

The selection of the reference model. We take Qwen3-32B as the reference model as it sits in the middle of the APIs pool - it is neither a cheap weak model nor a flagship but expensive one. The constraints of JOVE are set from unconstrained Qwen3-32B direct calls on Bamboogle, a relatively short multi-hop set with lower cost and latency than MMLU-Pro or GPQA. Direct Qwen3-32B on Bamboogle costs 2.00×10^{-4} USD and takes 14.09 s on average (Table 1); we therefore use $\gamma = 2.0 \times 10^{-4}$ USD/query, $\mu_t = 16$ s. The 16 s deadline is the Bamboogle direct latency with a small slack. Calibrating on this easier set makes the same (γ, μ_t) *tighter* than an average of direct Qwen3-32B cost and latency over all four datasets, where MMLU-Pro and GPQA are several times more expensive and slower.

Although JOVE may assign larger models from the pool, each such call is charged in full: under γ set to midsize spend on a cheap dataset, a 70B/120B assignment on one task node must be offset by cheaper assignments on other nodes and queries. Therefore, matching accuracy at this operating point is not a free upgrade to stronger LLMs.

We do not pass an explicit thinking/reasoning flag for Qwen3-32B. OpenRouter lists thinking as optional for this model. Downstream providers (DeepInfra, SiliconFlow) may turn thinking on for some queries depending on the difficulty levels. That behavior, when it occurs, applies to both the 32B baselines and JOVE.

A.4.1 DISCUSSION ON THE LEARNING SEQUENCE

Table 2: Results on MMLU-Pro. Each query generates one task graph in the learning sequence.

Methods	Accuracy (%) ↑	Latency (sec.) ↓	Cost ($\times 10^{-4}$ USD) ↓
Inference-level baselines (unconstrained)			
Direct (Qwen3-32B)	47.3 ± 0.8	45.39 ± 2.29	5.34 ± 0.38
CoT (Wei et al., 2022)	54.1 ± 1.1	34.11 ± 4.63	5.38 ± 0.34
SoT (Ning et al., 2024)	52.0 ± 3.1	70.17 ± 6.27	27.53 ± 3.71
Plato (Jin et al., 2025a)	52.8 ± 3.2	104.20 ± 5.21	22.23 ± 2.02
Resource-aware methods (constrained)			
WR-Online (Li et al., 2026)	43.7 ± 4.1	22.16 ± 2.33	2.04 ± 0.04
Knapsack-ascend	39.1 ± 3.6	5.88 ± 0.82	1.70 ± 0.18
Knapsack-descend	42.3 ± 3.8	7.59 ± 1.26	1.74 ± 0.10
JOVE single-node	47.1 ± 13.1	2.88 ± 0.71	1.75 ± 0.13
JOVE	$\mathbf{55.8} \pm 0.9$	7.32 ± 0.86	2.13 ± 0.32

As described in Section 4, we generate four task graphs per query, with 2, 3, 4, and 5 nodes. This yields 500 query–graph instances for Bamboogle, 48,000 for MMLU-Pro, 800 for LiveBench-Reasoning, and 1,792 for GPQA. We randomly shuffle these instances to form the online sequence. Varying the graph size exposes JOVE to different task structures and provides more decision rounds for learning, particularly on smaller datasets like Bamboogle (only 125 queries). Different graph sizes can change how a query is decomposed, for example, a larger graph may contain finer-grained subtasks, yielding different task graphs (different nodes) for the same query. We notice that for simple queries (e.g., two or three nodes are enough to clearly decompose the problem), similar nodes may appear across the four generated task graphs. However, this does not create direct answer leakage: JOVE neither stores nor reuses executor outputs or ground-truth answers, and every graph is executed independently. Verification labels only update the shared task–model quality estimator $\hat{\theta}_t$. Thus, any benefit arises through feature-based generalization across related subtasks, which is the intended learning mechanism rather than memorization of previous answers.

Table 2 evaluates the learning sequence in which each MMLU-Pro query appears only once, with the planner choosing its graph size. The inference-level baselines retain their results from Table 1 because they process each query directly. JOVE achieves 55.8% accuracy, compared with 54.8% in the four-graph setting, with similar cost and latency. JOVE's slightly higher accuracy may reflect the planner's freedom to choose graph size. An appropriately sized graph can avoid overly broad subtasks that require expensive executors, as well as unnecessary nodes that consume budget and latency. The other resource-aware methods likewise show no clear accuracy decline relative to their reported variability. The result suggests that JOVE's performance on MMLU-Pro does not depend on encountering multiple graphs for the same query. We use four graph sizes in the main experiments to provide more learning rounds, particularly for smaller datasets; the one-graph experiment provides a check under a sequence of distinct queries.

A.4.2 SATISFACTION OF THE CONSTRAINTS

Deadline hit rate. JOVE has the latency constraint:

$$\Pr_{\pi}\Big(L_t \le \mu_t - d_t^{\text{plan}} \,\Big|\, \mathcal{H}_t, \mathbf{a}_t\Big) \ge 1 - \delta_t$$

which indicates the end-to-end latency (from the start of task graph generation, to the end of the task graph execution) should be less than the target deadline μ_t with probability at least $1 - \delta_t$.

For the main experiment (Table 1), we set $\mu_t = 16$s and $\delta_t = 0.1$ for all task graphs. Notice that JOVE supports query-specific deadlines, which users can set according to their latency needs. We use a common deadline across queries to provide a consistent latency target for evaluation.

Table 3 reports the average deadline hit rate over successive rounds of online learning. In the final interval, every dataset reaches at least 96.0%, exceeding the 90% hit rate target. MMLU-Pro,

LiveBench-Reasoning, and GPQA initially fall below the 90% target, potentially because limited call histories yield inaccurate node-level quantile estimates. With more queries executed, each LLM has more latency samples to construct its context-based latency quantiles (detailed in Section A.3.4), providing precise latency estimations for the per-query MILP to decide the executor allocations. We notice that across all intervals, LiveBench-Reasoning remains the lowest deadline hit rate among all datasets. This can be explained by the fact that LiveBench-Reasoning queries are longer, which can increase LLM API processing time and require models to reason more, yielding high variance of the execution time.

Table 3: Deadline hit rate (%) across successive intervals. The target is 90%.

Datasets	$t \in [1, 50]$	$t \in [51, 100]$	$t \in [101, 200]$	$t \in [201, \text{end}]$
Bamboogle	96.8 ± 4.0	98.2 ± 1.6	98.1 ± 2.1	98.6 ± 1.4
MMLU-Pro	75.6 ± 8.8	93.4 ± 3.6	95.8 ± 1.7	98.4 ± 0.4
LiveBench-Reasoning	56.9 ± 10.0	89.7 ± 4.3	94.7 ± 4.0	96.0 ± 2.1
GPQA	72.9 ± 12.2	91.6 ± 5.6	95.7 ± 1.3	98.7 ± 1.3

Table 4: Average cost ($\times 10^{-4}$USD) across successive intervals. $\gamma = 2 \times 10^{-4}$ USD.

Datasets	$t \in [1, 50]$	$t \in [51, 100]$	$t \in [101, 200]$	$t \in [201, \text{end}]$
Bamboogle	11.29 ± 3.3	1.83 ± 0.3	1.38 ± 0.2	0.98 ± 0.2
MMLU-Pro	23.09 ± 12.4	7.65 ± 4.1	3.24 ± 1.2	1.99 ± 0.1
LiveBench-Reasoning	6.15 ± 2.0	2.39 ± 1.3	1.97 ± 0.5	1.97 ± 0.4
GPQA	21.35 ± 4.6	8.74 ± 3.2	3.05 ± 1.0	1.66 ± 0.1

Average cost results. The long-term cost constraint is defined as

$$\limsup_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{\pi}[C_t(\mathbf{a}_t, \mathbf{h}_t)] \leq \gamma.$$

JOVE uses a dynamic Lagrange multiplier λ_t to respond to budget overspending. When cost exceeds the target, λ_t rises and makes expensive LLMs less attractive in subsequent decisions. The overall average costs in Table 1 slightly exceed $\gamma = 2 \times 10^{-4}$ USD per query because JOVE spends more in early rounds, when λ_t is still small. Table 4 shows how this behavior changes over time: interval average costs decline as JOVE adjusts its selections. These finite-horizon results illustrate JOVE's budget-control behavior, complementing the asymptotic budget guarantee established for the oracle policy in Proposition A.3.

In early rounds, sparse call histories can cause JOVE to underestimate costs or high-percentile latencies. Cost can also exceed the target while the multiplier, initialized at $\lambda_1 = 0$, adjusts to observed spending. As more calls are observed, the estimates improve: in our experiments, interval-average costs approach the budget, and aggregate deadline hit rates exceed the 90% target in the final interval. These results provide empirical support for JOVE's deadline control as its estimates improve; establishing a conditional guarantee for every query under estimated quantiles remains future work.

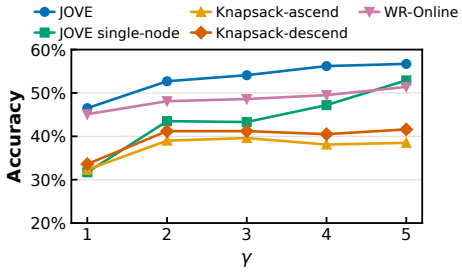
The paper's focus is the reduction of the graph-level latency requirement to linear MILP constraints and the joint allocation of execution and paid verification. The resource estimators are modular and can be replaced by offline-trained or more advanced predictors. For example, offline profiling could provide better estimates at the start of learning.

Feasibility in the analysis and implementation. Our analysis characterizes resource feasibility and quality learning through complementary oracle and online settings. For the oracle budget argument in Proposition A.3, we require a stationary policy that selects allocations in $\mathcal{F}_t(\mathbf{d}_t)$ while keeping expected total cost, including planning, execution, and verification, at most γ. This condition places the comparison policy within the same conservative node-quantile constraints used by the oracle MILP, ensuring that it is available to the optimizer. For online quality learning, Assumption 3.4 requires $\mathcal{F}_t(\hat{\mathbf{d}}_t)$ to be nonempty at each round and the online MILP to be solved to

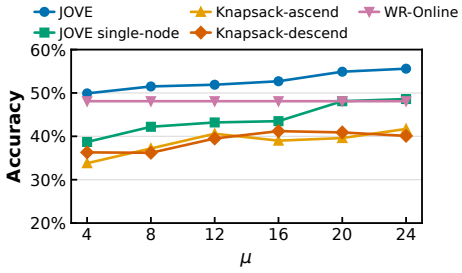
optimality. Giving the comparator the same resource estimates and budget prices as JOVE allows us to isolate the contribution of quality learning. We acknowledge that feasibility under estimated resource coefficients alone does not establish the true latency chance constraint or long-term budget feasibility; extending the oracle resource guarantees to the implemented policy requires additional control of estimation error. In particular, the latency argument in Appendix A.2.1 applies whenever the online MILP is feasible and its estimated node durations upper-bound the corresponding true conditional latency quantiles at the assigned risk levels. We complement these analytical results with empirical measurements of budget use and deadline hit rates, providing evidence of resource control in the evaluated workloads and motivating further analysis of online resource estimation.

A.4.3 Additional ablations

Ablation on cost and latency constraints. Figure 4 examines how the resource constraints affect JOVE's final-answer accuracy. As shown in Figure 4a, increasing the cost budget generally improves accuracy because JOVE can select stronger, more expensive LLMs and invoke verification more frequently. Similarly, Figure 4b shows that a more relaxed latency deadline expands the set of feasible allocations, allowing JOVE to use slower but more capable LLMs. Under tighter deadlines, it instead favors faster executors, resulting in a modest reduction in accuracy. Overall, these results demonstrate that the cost budget and latency deadline provide direct and interpretable controls over the accuracy–resource trade-off.



(a) Final-answer accuracy under different API-cost budgets γ ($\times 10^{-4}$USD).



(b) Final-answer accuracy under different latency deadlines (sec.).

Figure 4: Ablation of the cost and latency constraints on GPQA.

A.4.4 Comparison with random verification

To isolate the benefit of the proposed verification strategy (Remark 3.2), we compare JOVE with a random-verification baseline. This baseline uses the same per-query MILP for executor allocation but removes the information-gain bonus from the objective and selects each node for verification with probability p. Both methods use the same cost budget and latency constraint.

Table 5: Comparison of verification strategies on Bamboogle.

Methods	Accuracy(%)	Latency (s)	Cost ($\times 10^{-4}$USD)
$p = 0.2$	33.6 ± 1.7	4.23 ± 0.83	2.06 ± 0.11
$p = 0.5$	34.8 ± 1.2	3.27 ± 0.35	2.42 ± 0.24
$p = 0.8$	42.3 ± 0.9	3.09 ± 0.72	2.35 ± 0.14
$p = 0.9$	37.1 ± 1.4	3.02 ± 0.40	2.37 ± 0.12
$p = 1.0$	38.5 ± 1.2	4.00 ± 0.39	2.09 ± 0.09
JOVE	49.3 ± 1.0	3.68 ± 0.48	2.18 ± 0.18

Table 5 shows that JOVE outperforms random verification across different probabilities, improving accuracy by 7.0 percentage points over the best random baseline ($p = 0.8$). For the random baseline, low verification probabilities provide limited feedback, which can slow quality learning; frequent verification can consume too much budget that would otherwise support more capable executors,

potentially explaining the lower accuracy at $p = 0.9$ and $p = 1.0$. These results support selecting verification based on its information gain and cost rather than using a fixed probability.

Verifier reliability and the theory–experiment gap. Our benchmarks provide reference answers for complete queries (final responses), but not correctness labels for the intermediate subtasks generated by the planner. Therefore, we do not have the ground truth of the node outputs. We acknowledge that the conditionally unbiased feedback assumed in Assumption 3.3 is not established for the experimental verifier. This assumption allows verification noise to average out rather than systematically shift the estimated quality, supporting the confidence bounds around true expected correctness used in the regret proof. Figure 3a illustrates the practical setting: even when all five verified nodes are labeled correct, final-answer accuracy is 87.3%. This gap is consistent with verifier noise or imperfect alignment between node-level success criteria and final-answer correctness, rather than a direct estimate of label error rates. Nevertheless, final-answer accuracy decreases as the number of verifier-labeled incorrect nodes increases, suggesting that the feedback carries useful quality information. Together with JOVE's competitive final-answer accuracy and its advantage over random verification in Table 5, these results provide empirical support that learning remains effective with the available, potentially imperfect feedback in the evaluated workloads.

A.5 Enhanced JOVE

A.5.1 Joint optimization of node risk allocation

The main paper's formulation fixes node risk shares before optimizing the allocation. We extend it to select these shares jointly with the executor and verification decisions. This allows nodes on bottleneck paths to receive larger tolerances, reducing their required latency quantiles, while nodes on branches with slack can accommodate smaller tolerances. All paths remain subject to the response deadline.

Discrete risk levels. Before allocation, choose a small grid of risks

$$\mathcal{Q}_t = \{r_{t,1}, \ldots, r_{t,K}\} \subset (0, \delta_t].$$

These grid levels $r_{t,k}$ can be selected from breakpoints of the empirical node-level quantile records, which are piecewise constant in the risk tolerance. For each node, executor, and risk level, we have the latency quantile

$$\ell_t^i(a, r_{t,k}) = \inf\left\{d \geq 0 : \Pr\left(D_t^i(a) \leq d | \mathcal{H}_t\right) \geq 1 - r_{t,k}\right\}.$$

In the online implementation, these coefficients are replaced by $\hat{\ell}_t^i(a, r_{t,k})$, using the same estimator and observation histories as the main algorithm.

MILP formulation. To optimize the allocation of risk levels across the graph, we replace $z_t^i(a, \eta)$ with binary variables $x_t^i(a, \eta, k)$, where $x_t^i(a, \eta, k) = 1$ selects executor a, verification decision η, and risk level $r_{t,k}$ at node i. The original allocation and the selected risk share are recovered as

$$z_t^i(a, \eta) = \sum_{k=1}^{K} x_t^i(a, \eta, k), \qquad \delta_t^i = \sum_{a,\eta,k} r_{t,k} x_t^i(a, \eta, k).$$

Let $\mathbf{x}_t = \{x_t^i(a, \eta, k)\}_{a \in \mathcal{A}_t^i, \eta \in \{0,1\}, k=1,\ldots,K}$. Define the allocated node duration as

$$v_t^i(\mathbf{x}_t) = \sum_{a \in \mathcal{A}_t^i} \sum_{\eta \in \{0,1\}} \sum_{k=1}^{K} \ell_t^i(a, r_{t,k}) x_t^i(a, \eta, k).$$

The extended problem is

$$
\begin{aligned}
\max_{\mathbf{x}_t,\mathbf{F}_t} \quad & \sum_{i\in\mathcal{N}_t}\sum_{a\in\mathcal{A}_t^i}\sum_{\eta\in\{0,1\}}\sum_{k=1}^{K}\left[\frac{1}{|\mathcal{N}_t|}(\mathrm{UCB}_t^i(a)+k_v\eta I_t^i(a))-\lambda_t\hat{c}_t^i(a,\eta)\right]x_t^i(a,\eta,k)\\
\text{s.t.} \quad & \sum_{a,\eta,k} x_t^i(a,\eta,k)=1, \qquad \forall i\in\mathcal{N}_t,\\
& \sum_{i,a,\eta,k} r_{t,k}x_t^i(a,\eta,k)\le\delta_t,\\
& F_t^i\ge v_t^i(x), \qquad \forall i\in\mathcal{S}_t,\\
& F_t^i\ge F_t^j+v_t^i(x), \qquad \forall(j,i)\in\mathcal{E}_t,\\
& F_t^i\le\mu_t-d_t^{\mathrm{plan}}, \qquad \forall i\in\mathcal{T}_t,\\
& F_t^i\ge 0, \qquad \forall i\in\mathcal{N}_t,\\
& x_t^i(a,\eta,k)\in\{0,1\}, \qquad \forall i,a,\eta,k.
\end{aligned}
\tag{26}
$$

All objective and constraint coefficients are fixed before solving, so Eq. 26 remains a MILP, which is solvable using the same procedure as JOVE. The risk grid increases the number of binary variables by a factor of K and adds one global risk-budget constraint.

This extended version of JOVE incorporates node-level latency risk allocation directly into the per-query MILP. Compared with the uniform allocation of δ_t^i used in Section 3.2, optimizing the node-level risks inside the MILP generally yields tighter (less conservative) latency bounds, because the risk budget is concentrated on the critical path according to the per-node latency estimates. For instance, if one source–sink path contains a single node while another contains four, there is little benefit to spending risk budget on the single-node path; the budget is better spent loosening the bounds of the nodes that actually determine end-to-end latency.

A.6 NOTATION

Tables 6 and 7 collect the symbols used throughout the paper, grouped by role: the query and its task graph, the allocation decisions, the stochastic outcomes of execution and verification, the oracle statistics of Section 3.2, their online estimates from Section 3.3, the resource constraints and control parameters, and the constants appearing in the analysis of Section 3.4. Superscript i indexes a node, subscript t indexes a query, and a denotes a candidate executor. A bar $\bar{(\cdot)}$ marks an expected or oracle quantity and a hat $\hat{(\cdot)}$ marks its online estimate.

Table 6: Notation for the task graph, decisions, outcomes, and oracle statistics.

Symbol	Description
Query and task graph	
t, T	Query index and horizon (number of queries).
x_t	The t-th query.
$\mathcal{G}_t = (\mathcal{N}_t, \mathcal{E}_t)$	Planner-generated task graph (DAG) for query x_t.
$\mathcal{N}_t$, $\lvert\mathcal{N}_t\rvert$	Node (subtask) set and its cardinality.
$\mathcal{E}_t$	Edge set; $(j, i) \in \mathcal{E}_t$ means node i requires the output of node j.
$\mathcal{S}_t, \mathcal{T}_t$	Source and sink nodes of $\mathcal{G}_t$.
$\mathcal{P}(\mathcal{G}_t)$	Set of source-to-sink paths in $\mathcal{G}_t$.
$\mathcal{A}_t^i$	Candidate executor set at node i, chosen by the planner.
π	Online allocation policy.
$\mathcal{H}_t$	Information available before allocating query x_t.
Allocation decisions	
a_t^i, $\mathbf{a}_t$	Executor selected at node i; the vector of executors for query t.
$h_t^i \in \{0,1\}$, $\mathbf{h}_t$	Verification indicator at node i; the vector of verification decisions.
$\eta \in \{0,1\}$	Verification value in the joint encoding.
$z_t^i(a,\eta)$, $\mathbf{z}_t$	Binary indicator that node i uses LLM a with verification decision η.
F_t^i	Finish time of node i, relative to the start of graph execution.
$v_t^i(\mathbf{z}_t^i)$	Latency quantile selected at node i by the allocation.
$\mathcal{F}_t(\mathbf{d}_t)$	Feasible set of assignment and finish-time variables induced by node durations $\mathbf{d}_t$.
Stochastic outcomes (uppercase) and their realizations (lowercase)	
$Y_t^i(a) \in \{0,1\}$	Latent correctness of node i executed by model a; not observed from execution.
$\widehat{Y}_t^i \in \{0,1\}$	Binary label returned by the fixed verifier when $h_t^i = 1$.
$M_t^i(a)$, $m_t^i(a)$	Stochastic and realized API cost of executing node i with model a.
$\widetilde{M}_t^i$, $\widetilde{m}_t^i$	Stochastic and realized verification cost at node i.
$D_t^i(a)$, $d_t^i(a)$	Stochastic and realized execution latency of node i under model a.
M_t^{plan}, m_t^{plan}	Stochastic and realized API cost of task-graph generation (0 for a local planner).
d_t^{plan}	Graph-generation plus the time reserved for allocation.
L_t	Graph execution latency, equal to the critical-path duration.
$C_t(\mathbf{a}_t, \mathbf{h}_t)$, c_t	Total and realized API cost of processing query x_t (planning, execution, verification).
Oracle statistics (Section 3.2)	
$\bar{y}_t^i(a)$	Expected node correctness, $\mathbb{E}[Y_t^i(a)]$.
$\bar{c}_t^i(a,\eta)$	Expected execution plus verification cost at node i.
$\ell_t^i(a,\epsilon)$	Smallest latency met with probability at least $1-\epsilon$.
$\bar{d}_t^i(a) = \ell_t^i(a,\delta_t^i)$, $\mathbf{d}_t$	Node-level latency quantile at risk share δ_t^i; the collection over all (i,a).

Table 7: Notation for online learning, resource control, and the analysis.

Symbol	Description
Online estimation (Section 3.3)	
$\phi_t^i(a) \in \mathbb{R}^d$, d	Feature vector of the task–model pair and its dimension.
$\mathcal{D}_{t-1}$	Feature–label pairs obtained through verification before query t.
$\Upsilon_{t-1}(a)$	Execution records (ϕ, m, d) for model a collected before query t.
$w(\phi, \phi_t^i(a))$	Softmax similarity weight assigned to a past record.
G_{t-1}, G_{t-1}^+	Ridge information matrix before query t, and after adding one verified observation.
$\hat{\theta}_t$	Ridge-regression estimate of the quality parameter.
$u_t^i(a)$	Uncertainty score, $\|\phi_t^i(a)\|_{G_{t-1}^{-1}}$.
$\mathrm{UCB}_t^i(a)$	Optimistic quality estimate, clipped to $[0, 1]$.
$I_t^i(a)$	D-optimal information gain from verifying (i, a).
$\hat{m}_t^i(a)$, $\hat{\widetilde{m}}_t^i$	Estimated execution and verification cost.
$\hat{\ell}_t^i(a, \epsilon)$, $\hat{d}_t^i(a)$, $\hat{\mathbf{d}}_t$	Estimated latency quantile, node duration at risk δ_t^i, and their collection.
$\hat{c}_t^i(a, \eta)$	Estimated execution plus verification cost, $\hat{m}_t^i(a) + \eta \hat{\widetilde{m}}_t^i$.
$n_{t-1}(a)$, $\rho_{t-1}(a)$	Size of the retained history for model a and the corresponding baseline weight.
$m_0(a)$, $\ell_0(a, \epsilon)$	Model-specific baseline cost and latency quantile used before observations accumulate.
Resource constraints and control parameters	
γ	Long-run average API-cost budget per query.
μ_t	Response deadline for query t.
δ_t, δ_t^i	Latency violation tolerance for query t and the risk share assigned to node i.
λ_t, α_λ	Budget shadow price and its step size.
λ_{reg}	Ridge regularization parameter.
β_t	Confidence radius scaling the uncertainty bonus.
$k_v \geq 0$	Verification weight on the information-gain bonus.
Analysis (Section 3.4 and Appendix A.2.3)	
$\theta^\star$	True quality parameter, $\bar{y}_t^i(a) = \phi_t^i(a)^\top \theta^\star$.
L, S	Bounds on $\|\phi_t^i(a)\|_2$ and $\|\theta^\star\|_2$.
ε_t^i, R	Verification label noise and its sub-Gaussian parameter.
δ_{conf}	Failure probability of the confidence event.
$n_{\max}$, $\lambda_{\max}$, $\widetilde{m}_{\max}$	Bounds on nodes per query, budget price, and estimated verification cost.
$N = T n_{\max}$	Upper bound on the total number of executed nodes over the horizon.
$c_v = \lambda_{\max} \widetilde{m}_{\max}$	Upper bound on the priced verification cost.
C_{bat}, Γ_T	Batching constant and cumulative-uncertainty bound.
τ_T, K_T	Uncertainty threshold and lower endpoint of the admissible verification-weight interval.
$\mathcal{V}_t$, b_t	Nodes selected for verification at query t and their number.
$J_t(\mathbf{z}_t)$, $\mathbf{z}_t^\star$	Comparator objective with true qualities, and its maximizer.
$\mathcal{R}_T$	Quality-learning regret, $\sum_{t=1}^T [J_t(\mathbf{z}_t^\star) - J_t(\mathbf{z}_t)]$.