The Orchestrator Bottleneck: Formal Coordination Strategies for Cost-Optimal Multi-Agent Enterprise Workflows

Rudrendu Kumar Paul [1] **Sourav Nandy** [2]

Abstract

Multi-agent LLM systems increasingly depend on orchestrator agents to coordinate specialist agents, yet teams choose orchestration strategies by convention at system design time, rarely revisiting them as task distributions shift. We formalize the orchestrator as a constrained optimization problem over a typed agent registry with capability declarations, cost profiles, and trust boundaries. We define three coordination strategies, sequential delegation, parallel fan-out, and hierarchical decomposition, and derive threshold conditions under which each minimizes a combined cost-latency objective for a given task dependency structure. We implement ORCHESTRATE, a strategy-switching orchestrator that selects coordination mode dynamically via dependency graph analysis, and evaluate it on three enterprise workflow benchmarks: document processing, compliance review, and incident triage. On these benchmarks, ORCHESTRATE reduces end-to-end cost by 28–42% relative to static sequential baselines while trading at most 0.6 percentage points of task-completion quality. The cost gains come with two caveats: on highly parallelizable tasks, a static parallel baseline still achieves the lowest raw latency because it avoids strategy-selection overhead, and we identify an empirical orchestrator overhead threshold at 15% of total token budget beyond which coordination cost erodes the planning advantage entirely.

[1]Boston University, Boston, MA, USA [2]University of Texas at Austin, Austin, TX, USA. Correspondence to: Rudrendu Kumar Paul <rudrendupaul2022@gmail.com>, Sourav Nandy <sourav.nandy@gmail.com>.

Workshop on Statistical Frameworks for Uncertainty in Agentic Systems (AgenticUQ @ ICML 2026), Seoul, South Korea. 2026.
Copyright 2026 by the author(s).

1. Introduction

The compound AI systems paradigm (Zaharia et al., 2024) shifted production architectures from monolithic model serving toward pipelines of retrievers, generators, tool-callers, and validators. When those components are themselves LLM-powered agents, a coordinator must decide which agent handles which subtask, in what order, and with what resource budget. In frameworks such as AutoGen (Wu et al., 2023) and MetaGPT (Hong et al., 2024), this coordinator is the *orchestrator*: a distinguished agent responsible for task decomposition, delegation, and result aggregation.

Most production teams choose an orchestration strategy once during system design and never revisit it. A document-processing pipeline uses sequential handoffs because the first team that built it found sequential easy to debug. An incident-response system fans out to three specialist agents in parallel because the latency target demanded it. Neither team formally evaluated whether their strategy was cost-optimal for the task structure at hand, and when the task distribution shifted (longer documents, more ambiguous incident reports), the strategy stayed fixed.

This paper asks a direct question: given a set of typed agents with known cost profiles and a task with a dependency structure, which coordination strategy minimizes a weighted sum of token cost and wall-clock latency? We provide formal answers. Selecting among sequential, parallel, and hierarchical execution strategies is fundamentally a planning problem under cost and latency constraints: the orchestrator must construct a plan over agent actions before any subtask executes, making strategy selection a first-class planning decision.

Our contributions are:

1. A formal model of the orchestrator as a constrained planner over typed agent registries (Section 3), defining cost and latency as functions of task dependency graphs.
2. Closed-form conditions under which sequential delegation, parallel fan-out, and hierarchical decomposition are each optimal (Section 4).
3. ORCHESTRATE, a strategy-switching orchestrator that selects coordination mode at runtime via lightweight dependency graph analysis (Section 5).

4. Empirical evaluation on three enterprise benchmarks showing 28–42% cost reduction versus static baselines, with an honest accounting of where our approach loses to simpler strategies (Section 6).
5. Identification of the orchestrator overhead threshold: the point at which coordination cost itself becomes the bottleneck (Section 7).

2. Background and Related Work

Multi-agent coordination frameworks. Wu et al. (2023) introduced AutoGen, a conversational multi-agent framework where agents exchange messages in flexible topologies. Hong et al. (2024) structured collaboration through standardized operating procedures in MetaGPT, encoding human-style workflows as agent interaction protocols. Both frameworks fix the coordination topology at design time. Tran et al. (2025) surveyed collaboration mechanisms across five dimensions (actors, types, structures, strategies, protocols), with centralized orchestration appearing as one of several coordination topologies described. Their taxonomy characterizes coordination topologies without accounting for orchestration cost.

Adaptive and evolving orchestration. Dang et al. (2025) proposed a "puppeteer" paradigm where a centralized orchestrator trained via reinforcement learning dynamically directs agents based on evolving task states. Their key finding, that compact cyclic reasoning structures emerge under orchestrator evolution, motivates our work. However, their approach requires RL training, treating token cost as outside scope. Su et al. (2025) introduced Difficulty-Aware Agentic Orchestration (DAAO) that adjusts workflow depth per query using a variational autoencoder (VAE), a generative neural architecture that encodes task inputs into a continuous latent space and samples a difficulty score from the learned distribution. The VAE maps each incoming query to a scalar difficulty estimate, which then determines how many orchestration layers the system invokes. DAAO adjusts workflow depth and operator allocation per query; the strategy-selection problem we formalize here (choosing among sequential, parallel, and hierarchical coordination) falls outside DAAO's scope. Yu (2025) demonstrated that topology-aware orchestration achieves 12–23% improvement over static baselines and proposed a scaling law for orchestration advantage, though without a formal cost model.

Cost modeling and resource bounds. Ye & Tan (2026) formalized agent contracts with conservation laws for hierarchical delegation, achieving 90% token reduction and 525× lower variance. Their contract framework addresses resource bounds but not latency or strategy selection. Amin (2026) proposed Bayesian orchestration for cost-aware sequential decision-making, treating LLMs as likelihood models and aggregating via Bayes' rule, reducing costs by 34%. Ong et al. (2025) developed RouteLLM for dynamic model selection, reducing costs by 2× without quality loss, but routing selects *models*; we select *coordination strategies*. Kulkarni & Kulkarni (2025) benchmarked four orchestration architectures on 10K SEC filings and found hierarchical supervision occupies the best cost-accuracy Pareto position (F1 0.921 at 1.4× cost), with parallel fan-out being least token-efficient at 2.43%. Their work is closest to ours empirically, but offers no formal framework for predicting which strategy suits a given task.

Connection to planning. Our formalization shares structure with hierarchical task network (HTN) planning (Erol et al., 1994), where a high-level task is decomposed into subtasks via methods that respect ordering constraints. The orchestrator's strategy selection is a form of meta-level planning: choosing the decomposition method (sequential, parallel, hierarchical) that minimizes a cost-latency objective given the task's dependency graph. Classical HTN planners optimize plan length or makespan; we optimize token cost and wall-clock latency under LLM-specific cost models. This connection positions multi-agent orchestration as an instance of cost-aware HTN method selection, bridging the planning and multi-agent LLM literatures.

Gap. Existing work provides empirical benchmarks without formal strategy selection models, formal resource frameworks without cost-latency tradeoff analysis, or adaptive orchestrators without closed-form optimality conditions. No prior work formalizes when and why to switch coordination strategies based on task dependency structure.

3. Formal Model

Definition 1 (Agent Registry)**.** *An agent registry $\mathcal{R} = \{a_1, \ldots, a_n\}$ is a set of typed agents where each a_i is characterized by a tuple $(c_i, \tau_i, \kappa_i, \mathcal{T}_i)$: token cost per invocation $c_i \in \mathbb{R}^+$, expected latency $\tau_i \in \mathbb{R}^+$, capability set $\kappa_i \subseteq \mathcal{K}$ over a capability space $\mathcal{K}$, and trust boundary $\mathcal{T}_i$ specifying which other agents may receive its outputs.*

Definition 2 (Task Dependency Graph)**.** *A task T decomposes into a directed acyclic graph $G = (V, E)$ where each vertex $v_j \in V$ is a subtask requiring capability $k_j \in \mathcal{K}$, and each edge $(v_j, v_k) \in E$ indicates that v_k depends on the output of v_j. We define the dependency depth $d(G) =$ length of the longest path in G, and the parallelism width $w(G) =$ maximum antichain size.*

Definition 3 (Assignment Function)**.** *An assignment function $\phi : V \to \mathcal{R}$ maps each subtask $v_j \in V$ to an agent $a_i \in \mathcal{R}$, subject to the constraint that the assigned agent possesses the required capability: $k_j \in \kappa_{\phi(v_j)}$, where k_j is the capability required by subtask v_j and $\kappa_{\phi(v_j)}$ is the capability set of the assigned agent. Different assignments*

ϕ yield different costs because agents have different per-invocation costs c_i and latencies τ_i.

Definition 4 (Orchestration Cost)**.** *Given an assignment ϕ and task graph G, the total token cost $C(\phi, G)$ is the sum of three components:*

$$C(\phi, G) = \underbrace{c_{orch} \cdot |V|}_{overhead} + \underbrace{\sum_{v_j \in V} c_{\phi(v_j)}}_{execution} + \underbrace{\sum_{(v_j, v_k) \in E} c_{msg}(v_j, v_k)}_{messaging} \tag{1}$$

The first term is the orchestrator's own overhead: prompt tokens consumed for delegation, context passing, and result parsing, incurred once per subtask (c_{orch} is the per-subtask orchestrator cost, measured empirically). The second sums the execution cost of each agent $\phi(v_j)$ assigned to subtask v_j, where $c_{\phi(v_j)}$ is the token cost from Table 1. The third sums the inter-agent message cost c_{msg} for every dependency edge, covering tokens spent serializing one agent's output into another agent's input context. The orchestrator's objective is to find the assignment ϕ^* that minimizes $C(\phi, G)$ subject to the capability constraints from Definition 3 and the trust boundary constraints from Definition 1.

The total latency depends on the coordination strategy. We model three strategies in Section 4.

4. Coordination Strategies

We analyze three strategies under a combined objective $J = \alpha \cdot C + (1 - \alpha) \cdot L$ where C is normalized token cost and L is normalized latency, with $\alpha \in [0, 1]$ controlling the tradeoff.

Each strategy below defines its own cost and latency expression as a direct function of the dependency graph G and assignment ϕ. These expressions follow from the structure of each strategy: sequential delegation sums all agent latencies in series, parallel fan-out takes the maximum over concurrent agents, and hierarchical decomposition sums level-wise maxima across the hierarchy. The propositions are derived by setting $J_{\text{strategy}_1} = J_{\text{strategy}_2}$ for each pair of strategies and solving for the tradeoff weight α at which one strategy overtakes the other. The resulting threshold values tell practitioners when to switch coordination modes. The thresholds are parameterized by empirical quantities (R for redundancy, η for estimation accuracy, $w(G)$ for parallelism width) that must be estimated from deployment data. We report those values in Section 6 and provide full proof sketches in Appendix A.

4.1. Sequential Delegation

The orchestrator processes subtasks in topological order of G, delegating each to a single agent before proceeding to the next.

$$L_{\text{seq}}(\phi, G) = \sum_{v_j \in V} (\tau_{\text{orch}} + \tau_{\phi(v_j)}) \tag{2}$$

$$C_{\text{seq}}(\phi, G) = c_{\text{orch}} \cdot |V| + \sum_{v_j \in V} c_{\phi(v_j)} \tag{3}$$

Sequential delegation eliminates redundant computation entirely: no subtask is processed more than once. The token cost is minimal among strategies that preserve all dependency constraints. Note: the inter-agent messaging cost term $\sum_{(v_j, v_k) \in E} c_{\text{msg}}(v_j, v_k)$ from Equation 1 is absorbed into the per-subtask execution cost $c_{\phi(v_j)}$ in the strategy-specific expressions throughout this section, since each agent receives its predecessor's output in its input context window.

Proposition 1. *Sequential delegation minimizes J when $\alpha \geq \alpha^*_{seq}$, where*

$$\alpha^*_{seq} = \frac{w(G) - 1}{w(G) - 1 + R} \tag{4}$$

and R is the expected redundancy factor under parallel execution (fraction of tokens wasted on subtasks that are re-executed or discarded due to dependency violations).

The threshold α^*_{seq} is obtained by setting $J_{\text{seq}} = J_{\text{par}}$, substituting the cost and latency expressions from Equations 3–6 and 2–5, and solving for α. The numerator $w(G) - 1$ captures the latency advantage of parallelism (wider graphs benefit more), while the denominator adds R, the cost penalty from redundant computation. When cost matters more than latency (α is high) or the task is already mostly sequential ($w(G) \approx 1$), sequential delegation wins. In our enterprise benchmarks, compliance review workflows have $w(G) = 1.3$ on average, making sequential delegation optimal for $\alpha \geq 0.31$.

4.2. Parallel Fan-Out

The orchestrator dispatches all subtasks in the maximum antichain simultaneously, resolving dependencies via a merge step after parallel execution completes.

$$L_{\text{par}}(\phi, G) = \tau_{\text{orch}} + \max_{v_j \in V} \tau_{\phi(v_j)} + \tau_{\text{merge}} \tag{5}$$

$$C_{\text{par}}(\phi, G) = c_{\text{orch}} \cdot |V| + \sum_{v_j \in V} c_{\phi(v_j)} + R \cdot \sum_{v_j \in V_{\text{dep}}} c_{\phi(v_j)} \tag{6}$$

where $V_{\text{dep}} \subseteq V$ is the set of subtasks with unresolved dependencies at dispatch time, and R is the redundancy factor representing wasted tokens on subtasks that must be re-executed after the merge reveals dependency conflicts.

Proposition 2. *Parallel fan-out minimizes J when $\alpha \leq \alpha^*_{par}$, where*

$$\alpha^*_{par} = \frac{1}{1 + \frac{C_{par} - C_{seq}}{L_{seq} - L_{par}}} \tag{7}$$

This threshold follows from the same $J_{par} = J_{seq}$ equality, rearranged to isolate α as a function of the cost difference $(C_{par} - C_{seq})$ and latency difference $(L_{seq} - L_{par})$. The condition holds when the latency savings from parallelism outweigh the cost of redundant computation.

The redundancy factor R is the critical variable. When subtasks are truly independent ($|E| = 0$ within the antichain), $R = 0$ and parallel fan-out strictly dominates on latency at no additional token cost. In practice, hidden dependencies surface frequently. In our document-processing benchmark, 34% of subtask pairs flagged as independent by the orchestrator's initial analysis turned out to share context dependencies, pushing R to 0.41–0.58.

4.3. Hierarchical Decomposition

The orchestrator delegates to intermediate coordinators, each responsible for a subgraph of G. Each coordinator recursively applies either sequential or parallel execution within its subgraph.

$$L_{\text{hier}}(\phi, G) = \sum_{\ell=1}^{d_h} \left(\tau^{(\ell)}_{\text{coord}} + \max_{v_j \in V^{(\ell)}} \tau_{\phi(v_j)} \right) \tag{8}$$

$$C_{\text{hier}}(\phi, G) = \sum_{\ell=1}^{d_h} c^{(\ell)}_{\text{coord}} \cdot |V^{(\ell)}| + \sum_{v_j \in V} c_{\phi(v_j)} \tag{9}$$

where d_h is the hierarchy depth and $V^{(\ell)}$ are the subtasks at level ℓ.

Hierarchical decomposition adds coordination overhead at each level ℓ. The strategy pays off when G decomposes cleanly into weakly connected subgraphs, allowing parallel execution of subgraphs with sequential execution within them.

Proposition 3. *Hierarchical decomposition minimizes J when $\alpha^*_{par} < \alpha < \alpha^*_{seq}$ and the task complexity estimation accuracy η satisfies $\eta > \eta^* = 1 - \frac{c_{coord} \cdot d_h}{C_{par} - C_{seq}}$.*

The η^* threshold is derived by computing the expected cost penalty from misclassified subtasks: each misclassified subtask incurs the hierarchical coordination overhead c_{coord} without the corresponding planning benefit. Summing over d_h hierarchy levels and requiring the net cost to remain below C_{par} yields the bound. The dependence on η is the weak link. When the orchestrator's estimate of task structure is wrong, it creates hierarchies that either serialize parallelizable work or parallelize dependent work, losing the benefits of both alternatives. We measure η directly in Section 6.

Algorithm 1 ORCHESTRATE Strategy Selection

Require: Task T, Agent Registry $\mathcal{R}$, cost weight α
Ensure: Coordination strategy s^*, assignment ϕ^*
Parse T into dependency graph $G = (V, E)$ using LLM-based decomposition
Compute $d(G)$, $w(G)$, connected components $\{G_1, \ldots, G_m\}$
Estimate complexity $\hat{\eta}$ via calibrated classifier on $(d(G), w(G), |V|, |E|)$
Estimate redundancy $\hat{R}$ from historical dependency violation rate
Compute $\alpha^*_{\text{seq}}, \alpha^*_{\text{par}}$ from Propositions 1, 2
if $\alpha \geq \alpha^*_{\text{seq}}$ **then**
  $s^* \leftarrow$ SEQUENTIAL
else if $\alpha \leq \alpha^*_{\text{par}}$ **then**
  $s^* \leftarrow$ PARALLEL
else if $\hat{\eta} > \eta^*$ **then**
  $s^* \leftarrow$ HIERARCHICAL
else
  $s^* \leftarrow$ PARALLEL with post-hoc reconciliation
end if
Compute ϕ^* via capability matching on $\mathcal{R}$
return s^*, ϕ^*

5. ORCHESTRATE: Strategy-Switching Orchestration

Propositions 1–3 provide static decision boundaries: given known values of α, $w(G)$, R, and η, a practitioner can select the optimal strategy analytically. In practice, however, these quantities are not known a priori for an incoming task. They must be estimated at runtime from the task's dependency structure and from historical execution data. ORCHESTRATE bridges this gap by operationalizing the formal model in a three-phase pipeline that estimates the required parameters, applies the propositions' threshold conditions, and executes the selected strategy:

Phase 1: Dependency graph extraction. Given a natural-language task description, an LLM decomposes it into subtasks and infers dependencies. We use a constrained generation approach: the LLM outputs a JSON-formatted DAG where each node specifies a required capability from $\mathcal{K}$. Malformed or cyclic outputs trigger a single retry with explicit acyclicity instructions. In our benchmarks, first-attempt extraction succeeds 87% of the time; with one retry, 96%.

Phase 2: Strategy selection. Algorithm 1 selects the coordination strategy based on graph properties and the cost-latency weight α. The redundancy estimate $\hat{R}$ is maintained as an exponential moving average over the last 100 workflow executions, starting from a prior of 0.3, which reflects the empirical dependency violation rate observed across our

enterprise benchmark tasks during initial deployment. The complexity accuracy estimate $\hat{\eta}$ uses a lightweight gradient-boosted classifier trained on 2,400 labeled task decompositions from enterprise logs.

Phase 3: Execution and monitoring. During execution, ORCHESTRATE tracks actual token consumption and latency per subtask. If the running coordination overhead (orchestrator tokens / total tokens) exceeds 15%, the system triggers a mid-execution strategy switch to parallel fan-out with post-hoc reconciliation, bypassing further hierarchical coordination. This threshold emerges from our analysis in Section 7. Parallel fan-out is preferred over Sequential as the fallback because its latency ceiling is bounded by $\max_j(\tau_j)$ regardless of remaining graph depth, whereas Sequential cost scales with the remaining dependency path length, a quantity that is unreliable to re-estimate once mid-execution complexity has already exceeded the initial classifier prediction.

6. Experiments

6.1. Benchmarks

We evaluate on three enterprise workflow benchmarks constructed from anonymized production logs at two large enterprises:

- **DocProcess**: 1,200 document-processing workflows involving extraction, summarization, cross-referencing, and formatting across multi-section reports. Average $|V| = 6.8$, $d(G) = 3.2$, $w(G) = 2.4$.
- **ComplianceReview**: 800 regulatory compliance workflows requiring policy lookup, clause matching, risk scoring, and report generation. Average $|V| = 4.3$, $d(G) = 3.8$, $w(G) = 1.3$. Highly sequential.
- **IncidentTriage**: 600 IT incident response workflows involving log analysis, root-cause hypothesis generation, runbook retrieval, and remediation planning. Average $|V| = 5.1$, $d(G) = 2.1$, $w(G) = 3.4$. Highly parallelizable.

6.2. Agent Registry

The registry contains six agent types implemented with GPT-4o and Claude 3.5 Sonnet:

Agent Type	Cost (tokens)	Latency (s)
Extractor	2,100 $\pm$ 340	2.8 $\pm$ 0.6
Summarizer	3,400 $\pm$ 510	4.1 $\pm$ 0.9
Analyzer	4,200 $\pm$ 620	5.3 $\pm$ 1.2
Retriever	1,800 $\pm$ 280	1.9 $\pm$ 0.4
Validator	2,600 $\pm$ 390	3.2 $\pm$ 0.7
Generator	5,100 $\pm$ 780	6.7 $\pm$ 1.5

Table 1. Agent registry with mean token cost and latency ($\pm$ 1 standard deviation) measured over 500 invocations per agent type.

6.3. Baselines

We compare ORCHESTRATE against four baselines:

- **Static-Seq**: Fixed sequential delegation for all tasks.
- **Static-Par**: Fixed parallel fan-out for all tasks.
- **Static-Hier**: Fixed two-level hierarchical decomposition.
- **AdaptOrch** (Yu, 2025): Task-adaptive orchestration without formal cost modeling.

6.4. Results

ORCHESTRATE's primary advantage is cost reduction: it achieves the lowest token cost on all three benchmarks (Table 2), with 28–42% savings over Static-Seq. Latency and quality, by contrast, present genuine tradeoffs where simpler strategies outperform on specific metrics.

On DocProcess (moderate parallelism, $w(G) = 2.4$), ORCHESTRATE selects hierarchical decomposition for 61% of tasks and parallel fan-out for 29%, achieving 33% cost reduction over Static-Seq. However, Static-Par achieves lower latency (0.49$\times$ vs. 0.54$\times$) by parallelizing unconditionally, and Static-Seq retains the highest quality (87.4 vs. 86.8) because sequential processing avoids the dependency misclassification errors that affect strategy-switching systems. The 0.6-point quality drop occurs because 10% of tasks route through a suboptimal strategy due to misclassified dependency structures.

On ComplianceReview (low parallelism, $w(G) = 1.3$), ORCHESTRATE correctly identifies the sequential nature and uses sequential delegation for 78% of tasks. Cost drops 28% by avoiding the overhead of unnecessary hierarchical coordination. Quality is within 0.3 points of Static-Seq. Static-Par again achieves the best latency (0.73$\times$), though its cost is 41% higher than the baseline due to redundant parallel computation on this mostly-sequential workflow. AdaptOrch achieves similar quality but worse cost because it lacks a formal model for when hierarchical overhead exceeds its benefit.

On IncidentTriage (high parallelism, $w(G) = 3.4$), the tradeoff is most visible. Static-Par achieves the lowest latency (0.39$\times$) because it always parallelizes without strategy-selection overhead, and this is the benchmark where unconditional parallelism is most naturally suited. ORCHESTRATE incurs 0.43$\times$ latency, slightly worse, because it spends tokens analyzing the dependency graph before dispatching. The cost advantage (0.58$\times$ vs. 1.38$\times$ for Static-Par) is the largest across all benchmarks, driven by avoiding redundant parallel generation on the 19% of subtask pairs that harbor hidden dependencies. AdaptOrch edges out ORCHESTRATE on quality by 0.3 points, likely because its heuristic routing avoids the occasional misclassification

	DocProcess			ComplianceReview			IncidentTriage		
Method	Cost↓	Lat↓	Qual↑	Cost↓	Lat↓	Qual↑	Cost↓	Lat↓	Qual↑
Static-Seq	1.00	1.00	**87.4**	1.00	1.00	**91.2**	1.00	1.00	84.6
Static-Par	1.52	**0.49**	85.1	1.41	**0.73**	88.7	1.38	**0.39**	83.9
Static-Hier	1.21	0.68	86.9	1.18	0.82	90.5	1.15	0.59	84.2
AdaptOrch	0.88	0.71	87.1	0.91	0.89	90.8	0.84	0.63	**84.8**
ORCHESTRATE	**0.67**	0.54	86.8	**0.72**	0.78	90.9	**0.58**	0.43	84.5

Table 2. Results across three benchmarks. Cost and Latency are normalized to Static-Seq as the reference baseline (lower is better); accordingly, Static-Seq always shows 1.00 for these two columns. Quality is absolute task-completion accuracy (%, higher is better, not normalized). ORCHESTRATE uses $\alpha = 0.5$ (equal cost-latency weight). Bold marks the best value per column. ORCHESTRATE's primary advantage is cost reduction (best in all three benchmarks). Static-Par achieves the lowest latency across all benchmarks because it always parallelizes without strategy-selection overhead. Quality leaders vary by benchmark.

Task Type	Accuracy η	Strategy Error Rate
Well-structured	89.3%	4.2%
Semi-structured	72.1%	11.8%
Ambiguous NL	51.4%	23.6%
Overall	74.2%	11.4%

Table 3. Task complexity estimation accuracy by input type. Strategy error rate measures how often the wrong coordination strategy is selected due to incorrect complexity estimation.

Strategy	ρ (mean)	ρ (P95)
Sequential	8.2%	12.1%
Parallel	6.4%	9.8%
Hierarchical	14.7%	22.3%

Figure 1. Orchestrator overhead ratio ρ by strategy. Hierarchical decomposition incurs the highest overhead due to multi-level coordination prompts.

penalty our formal model incurs on ambiguous inputs.

Cost reduction is the consistent and primary advantage of ORCHESTRATE across all three benchmarks. Latency improvements are a secondary effect that depends on task structure; Static-Par achieves the lowest raw latency on every benchmark. Quality tradeoffs are small (0.1–0.7 percentage points) but real, driven by strategy misclassification.

6.5. Task Complexity Estimation Accuracy

The estimation challenge is sharpest for ambiguous inputs (Table 3). Well-structured tasks (API-formatted inputs with explicit subtask definitions) allow near-90% accuracy because the dependency structure is explicit. Ambiguous natural-language requests force the orchestrator to infer dependencies from semantic content, where accuracy drops to 51%. The 23.6% strategy error rate on ambiguous inputs is the primary source of suboptimality in ORCHESTRATE's performance, it occasionally applies hierarchical decomposition to tasks that would benefit from parallel fan-out, or vice versa.

This accuracy gap is not unique to our system. Kim et al. (2025) found that multi-agent systems degrade performance by 39–70% on sequential reasoning tasks when using parallelized architectures, precisely the failure mode caused by incorrect dependency estimation.

7. The Orchestrator Overhead Threshold

Every coordination strategy incurs orchestrator overhead: tokens consumed by task analysis, delegation prompts, progress tracking, and result aggregation, overhead that displaces actual subtask work. We define the overhead ratio $\rho = C_{\text{orch}}/C_{\text{total}}$.

Hierarchical decomposition averages 14.7% overhead, with the 95th percentile reaching 22.3% (Figure 1). When $\rho > 0.15$, the coordination cost erodes the benefits of intelligent strategy selection. At that threshold, the system's net cost exceeds what a flat parallel fan-out with a simple post-hoc reconciliation step would produce.

Why 15%? Consider a hierarchical workflow with $d_h = 3$ levels. Each level's coordinator consumes approximately $c_{\text{coord}} \approx 1{,}200$ tokens for decomposition, delegation, and aggregation. With five subtasks averaging 3,500 tokens each, the coordination budget is $3 \times 1{,}200 = 3{,}600$ tokens against $5 \times 3{,}500 = 17{,}500$ in subtask work, yielding $\rho = 0.171$. Adding a fourth level pushes ρ past 0.20, where the cost of coordination exceeds the savings from better task allocation. The $\rho = 0.171$ figure is illustrative of a moderate 3-level hierarchy, not a worst-case scenario. The mean overhead across all hierarchical workflows in our benchmark is 14.7% (Figure 1), reflecting a right-skewed distribution in which most workflows operate near but below the threshold; the 12% fallback trigger rate is driven by the upper tail of this distribution.

This threshold drives ORCHESTRATE's mid-execution fallback mechanism. When the running ρ crosses 0.15 during

a hierarchical workflow, the system abandons remaining hierarchical delegation, dispatches outstanding subtasks in parallel, and reconciles results with a single aggregation pass. In our benchmarks, this fallback triggers on 12% of hierarchical workflows and recovers an average of 2,100 tokens per affected workflow.

8. Discussion and Limitations

Limitations of the formal model. Our cost model assumes agents have stable per-invocation costs, which holds for API-based LLMs with fixed pricing but breaks down for self-hosted models where cost depends on batch utilization and GPU occupancy. The dependency graph formalization assumes tasks decompose into clean DAGs, but real enterprise workflows sometimes have feedback loops (e.g., a validator requesting re-generation from an earlier agent), requiring cycle-handling extensions.

Dependency estimation remains the weak link. The 51% accuracy on ambiguous natural-language inputs means ORCHESTRATE makes the wrong strategy choice nearly half the time for unstructured requests. Improving this requires either better task decomposition models or structured input formats that make dependencies explicit. In production, we found that adding a 200-token "dependency hint" to ambiguous requests (generated by a separate lightweight model) improved accuracy to 68%, but at additional cost.

Benchmark limitations. Our benchmarks are drawn from two companies and three workflow types. Financial services, healthcare, and other regulated industries may exhibit different dependency distributions. The benchmarks also assume a fixed agent registry; production systems where agents are added or removed dynamically would need online adaptation of the cost profiles.

Generalization beyond LLM agents. The formal model applies to any system with typed components, measurable costs, and dependency constraints. Classical service orchestration in microservice architectures faces analogous tradeoffs, and the optimality conditions from Section 4 could inform service mesh routing decisions.

9. Conclusion

We formalized the multi-agent orchestrator as a constrained planning problem and derived threshold conditions under which sequential, parallel, and hierarchical coordination strategies minimize a cost-latency objective. ORCHESTRATE translates these conditions into a practical strategy-switching system that reduces cost by 28–42% on enterprise workflows, though it does not dominate on every metric: static parallel baselines still achieve lower raw latency on highly parallelizable tasks, and strategy misclassification costs 0.3–0.7 quality points on some benchmarks. The orchestrator overhead threshold at roughly 15% provides a concrete decision boundary for practitioners: if your coordination tokens approach that fraction of total spend, simplify the coordination structure: more coordination overhead is the problem, not a feature to engineer around.

Impact Statement

Enterprise multi-agent workflows that reduce coordination cost by 28–42% have direct sustainability implications: lower token budgets per query aggregate to meaningful energy reductions at production scale. The formal cost model makes orchestration decisions auditable and reproducible rather than implicit in framework defaults, which supports responsible deployment practices where resource use is a first-class engineering concern. The orchestrator overhead threshold we identify at 15% gives practitioners a concrete decision criterion for when to simplify their coordination architecture rather than adding complexity. We identify no significant negative societal impacts from this work; the optimization framework is neutral with respect to the content of the workflows it coordinates, and safety properties of individual agents remain outside its scope.

References

Amin, D. Bayesian orchestration of multi-LLM agents for cost-aware sequential decision-making. *arXiv preprint arXiv:2601.01522*, 2026. URL https://arxiv.org/abs/2601.01522.

Dang, Y., Qian, C., Luo, X., Fan, J., Xie, Z., Shi, R., Chen, W., Yang, C., Che, X., Tian, Y., Xiong, X., Han, L., Liu, Z., and Sun, M. Multi-agent collaboration via evolving orchestration. *arXiv preprint arXiv:2505.19591*, 2025. URL https://arxiv.org/abs/2505.19591. Accepted at NeurIPS 2025.

Erol, K., Hendler, J., and Nau, D. S. HTN planning: Complexity and expressivity. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 1123–1128, 1994. URL https://www.cs.umd.edu/~nau/papers/erol1994htn.pdf.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.00352.

Kim, Y., Gu, K., Park, C., Park, C., Schmidgall, S., Heydari, A. A., Yan, Y., Zhang, Z., Zhuang, Y., Malho-

tra, M., Liang, P. P., Park, H. W., Yang, Y., Xu, X., Du, Y., Patel, S., Althoff, T., McDuff, D., and Liu, X. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025. URL https://arxiv.org/abs/2512.08296.

Kulkarni, S. and Kulkarni, Y. Benchmarking multi-agent LLM architectures for financial document processing: A comparative study of orchestration patterns, cost-accuracy tradeoffs and production scaling strategies. *arXiv preprint arXiv:2512.22651*, 2025. URL https://arxiv.org/abs/2512.22651.

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. RouteLLM: Learning to route LLMs with preference data. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2406.18665.

Su, J., Lan, Q., Xia, Y., Sun, L., Tian, W., Shi, T., Song, X., He, L., and Jingsong, Y. Difficulty-aware agentic orchestration for query-specific multi-agent workflows. *arXiv preprint arXiv:2509.11079*, 2025. URL https://arxiv.org/abs/2509.11079. Accepted at WWW 2026.

Tran, K.-T., Dao, D., Nguyen, M.-D., Pham, Q.-V., O'Sullivan, B., and Nguyen, H. D. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*, 2025. URL https://arxiv.org/abs/2501.06322.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

Ye, Q. and Tan, J. Agent contracts: A formal framework for resource-bounded autonomous AI systems. *arXiv preprint arXiv:2601.08815*, 2026. URL https://arxiv.org/abs/2601.08815. Accepted at COINE 2026.

Yu, G. AdaptOrch: Task-adaptive multi-agent orchestration in the era of LLM performance convergence. *arXiv preprint arXiv:2512.16873*, 2025. URL https://arxiv.org/abs/2512.16873.

Zaharia, M., Khattab, O., Chen, L., Davis, J. Q., Miller, H., Potts, C., Zou, J., Carbin, M., Frankle, J., Rao, N., and Ghodsi, A. The shift from models to compound AI systems. Berkeley Artificial Intelligence Research Blog, 2024. URL https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

A. Proof Sketches

Proposition 1. Sequential delegation incurs zero redundancy ($R = 0$) and latency $L_{\text{seq}} = \sum_j (\tau_{\text{orch}} + \tau_j)$. Parallel fan-out reduces latency to $L_{\text{par}} \approx L_{\text{seq}}/w(G)$ but adds redundancy cost $R \cdot \sum_{v_j \in V_{\text{dep}}} c_j$. Setting $J_{\text{seq}} \leq J_{\text{par}}$ and solving for α yields the threshold $\alpha^*_{\text{seq}} = (w(G) - 1)/(w(G) - 1 + R)$. When $w(G) = 1$ (fully sequential task), $\alpha^*_{\text{seq}} = 0$, meaning sequential delegation is optimal for all α, as expected.

Proposition 2. Follows from setting $J_{\text{par}} \leq J_{\text{seq}}$ and $J_{\text{par}} \leq J_{\text{hier}}$. The first inequality gives the threshold directly. The second requires $R < (C_{\text{hier}} - C_{\text{par}})/\sum_{v_j \in V_{\text{dep}}} c_j$, which holds when the redundancy from parallel execution is less than the coordination overhead from hierarchical coordination.

Proposition 3. Hierarchical decomposition lies between sequential and parallel on both cost and latency axes. It dominates both only when α falls in the intermediate range and the complexity estimate $\hat{\eta}$ is accurate enough that the hierarchical structure matches the true dependency graph. The accuracy threshold η^* follows from the requirement that misclassified subtasks do not push the effective cost above C_{par}, yielding $\eta^* = 1 - c_{\text{coord}} \cdot d_h/(C_{\text{par}} - C_{\text{seq}})$.

B. Reproducibility Details

Models. GPT-4o (gpt-4o-2024-05-13) and Claude 3.5 Sonnet (claude-3-5-sonnet-20241022). Temperature 0.0 for all agent invocations. Orchestrator uses GPT-4o.

Hardware. All experiments run via API calls. Client machine: Apple M2 Pro, 32GB RAM. Network latency to API endpoints: 45±12ms (measured over 1,000 pings).

Dependency graph extraction. Single-shot JSON generation with a 512-token system prompt specifying the DAG schema. Retry on parse failure or cycle detection, with explicit "ensure no cycles" instruction appended.

Complexity classifier. XGBoost with 4 features ($d(G)$, $w(G)$, $|V|$, $|E|$). Trained on 2,400 labeled decompositions (80/20 train/test split). Hyperparameters: max_depth=4, n_estimators=200, learning_rate=0.1.

Redundancy estimator. Exponential moving average with decay $\lambda = 0.95$ over the last 100 executions. Initial prior $\hat{R}_0 = 0.3$.

Cost normalization. All costs reported as fractions of the Static-Seq baseline for the same task. Absolute token counts are available in our supplementary materials.