Truthful and Cost-Minimizing Model Routing in Graph-Based Agentic Workflows

Anonymous ACL submission

Abstract

As Large Language Models (LLMs) evolve into modular agentic workflows, statically assigning frontier models to every sub-task becomes economically unsustainable. While dynamic routing offers a solution, existing approaches are often *myopic*: they overlook the substantial *context switching costs* incurred when transferring state between models, and they fail to account for the *strategic incentives* of self-interested agents in decentralized marketplaces. To address these challenges, we propose **STAR** (**S**trategy-proof **T**opology-aware **A**gent **R**outing), a mechanism that simultaneously minimizes execution costs and guarantees truthful bidding. STAR models the allocation problem as a reverse auction on a dependency graph. It employs a *virtual cost minimization* objective derived from Myerson's optimal auction theory to ensure dominant strategy incentive compatibility (DSIC), while rigorously internalizing the path-dependent externalities of context transfer. We evaluate STAR across four diverse workflow topologies (e.g., software development, legal analysis) under realistic market volatility. Results demonstrate that STAR reduces total payment by up to 33.6% compared to myopic baselines.

1 Introduction

The paradigm of LLM utilization is shifting from single-turn queries to complex modular agentic workflows. Frameworks such as Graph of Thoughts (Besta et al., 2024), MetaGPT (Hong et al., 2023), and ChatDev (Qian et al., 2024) demonstrate that decomposing a complex task into a directed graph of specialized sub-tasks significantly enhances performance. Crucially, in these "agentic workflows", different nodes may require distinct capabilities—some requiring high-level reasoning suitable for frontier models, while others involve routine generation tasks that can be handled by smaller, more cost-effective models.

Concurrently, the landscape of LLMs exhibits significant *performance heterogeneity* across downstream tasks. Empirical studies demonstrate that while massive general-purpose models are essential for complex reasoning, smaller, domain-specialized models often yield competitive or superior performance on specific verticals such as code generation or creative composition, typically at a fraction of the inference cost (Chen et al.; Ong et al., 2024; Zhang et al., 2025). Consequently, relying solely on a frontier model for every node in a workflow results in significant resource over-provisioning, rendering the system economically suboptimal.

As LLM providers diversify, we envision an open marketplace for operating agentic workflows. An orchestrator seeks to execute a task graph by procuring services from a pool of profit-maximizing third-party agents. A central challenge is *economical efficiency*: How can we minimize the total cost while ensuring that selected agents satisfy competency constraints for each node?

To understand the limitations of existing approaches, consider a two-step workflow: a Writer (v_1) generates a document, followed by an Editor (v_2) who refines it. Existing routers like FrugalGPT (Chen et al.) or LLM-Rec (Zhang et al., 2025) typically operate at a *single-task granularity*. They optimize the model selection for v_1 and v_2 *independently* based on local cost-performance trade-offs. While intuitively appealing, this myopic approach fails in a graph context due to two critical factors:

1. **Token-Based Pricing and Switching Costs:** Standard commercial LLM services charge based on token counts, including the *input tokens* (prompt processing and history) and the *output tokens* (text generation). While a naive router might switch agents to chase a lower output cost, it ignores the hidden *transfer overhead*. To maintain workflow conti-

nuity, switching to a new agent requires refeeding upstream information as new input tokens. This incurs a redundant input processing fee—effectively a "switching tax"—that can easily negate the savings from a cheaper generator.

2. **Dynamic Costs and Strategic Incentives:** Unlike fixed-price APIs, we envision an open marketplace where agents are third-party providers running on heterogeneous hardware. In this setting, an agent's internal cost per token is not static; it fluctuates based on real-time factors such as *GPU spot prices*, *electricity rates*, and *current server load*.
To minimize the total cost of the workflow, the system must induce *competition*, typically by soliciting bids from the agents and selecting the lowest offer. However, under a standard "pay-as-bid" model (where the winner is paid exactly their quoted price), competition forces agents to play a complex guessing game: they must inflate their bids above their costs to make a profit, attempting to price just below their competitors. This strategy is risky; if a low-cost agent overestimates the market price and inflates their bid too aggressively, they may lose to a higher-cost agent who bid more conservatively. Consequently, the submitted bids may no longer reflect the true underlying costs, preventing the system from minimizing the *final procurement payment*.
To resolve this, we employ a mechanism grounded in the Nobel Prize-winning theory of optimal mechanism design established by Myerson (1981). The key is to *decouple* an agent's bid from their potential payment. In our framework, an agent's bid determines only *whether* they are selected, while their payment is calculated entirely based on *competitors' bids*. Since an agent cannot influence the payment they receive by manipulating their own bid, their dominant strategy is to report their cost honestly.

In this paper, we propose STAR (**S**trategy-proof **T**opology-aware **A**gent **R**outing), a novel mechanism that simultaneously minimizes execution costs and guarantees truthful bidding for agentic workflows. We model the problem as a **reverse auction** over a directed graph. Unlike a traditional auction where buyers compete for an item, a reverse auction functions like a procurement system: the user acts as a single buyer soliciting services, while agents act as competitive sellers bidding to perform tasks at the lowest cost.

To achieve this, STAR operates not on raw reported costs, but on *virtual costs* (Myerson, 1981). Intuitively, a "virtual cost" transforms an agent's bid by adding a necessary *incentive markup*—this represents the extra margin the system must implicitly account for to make honesty the most profitable bidding strategy. By minimizing this adjusted cost metric, our framework allows us to find the globally optimal route while rigorously guaranteeing that agents have no incentive to manipulate prices, a property known as dominant strategy incentive compatibility (DSIC).

2 Related Work

Our work serves as a synthesis of three active research streams: structured agentic orchestration, economical model routing, and mechanism design for graph-structured dependencies.

Structured LLM Orchestration. Recent advancements have shifted from single-turn LLM interactions to complex, multi-step workflows. Structural prompting strategies like *Chain-of-Thought* (Wei et al., 2022), *Tree of Thoughts* (Yao et al., 2023), and *Graph of Thoughts* (Besta et al., 2024) demonstrate that imposing topological constraints on reasoning significantly improves performance. Frameworks such as *AutoGen* (Wu et al., 2024), *MetaGPT* (Hong et al., 2023), and *ChatDev* (Qian et al., 2024) further operationalize this by enabling arbitrary interaction graphs among specialized agents. However, these frameworks focus almost exclusively on *capability maximization* (i.e., "can we solve the task?"). They typically treat agents as fully aligned entities with no private incentives. We argue that this ignores the economic reality of open marketplaces. In our model, while agents remain *cooperative in executing the workflow*, they are *strategically self-interested regarding their compensation*. Unlike internal modules that simply obey, these independent providers aim to maximize their own profit, creating a disconnect between the user's goal (minimizing cost) and the agents' incentives (maximizing revenue).

Economical LLM Routing and Cascading. Addressing the high cost of LLM inference, a parallel line of research explores dynamic model selection. *FrugalGPT* (Chen et al.) proposes a cascading

architecture to sequentially query cheaper models. *RouteLLM* (Ong et al., 2024) and recent preference-based routers (Zhang et al., 2025) learn to dispatch queries to the most cost-effective provider based on predicted difficulty. While effective, these approaches operate under a **centralized optimization paradigm** with two critical limitations: (1) **Information Asymmetry:** They typically assume fixed, public pricing. However, we argue that an open LLM marketplace will inevitably mirror physical infrastructure realities, where token costs fluctuate with *electricity spot prices* and *real-time compute and data center demand*. In such a dynamic setting, providers possess private information about their instantaneous operating costs and have incentives to misreport them to maximize profit. (2) **Independence Assumption:** They typically route individual queries independently. They fail to model the switching cost ($S_{u,v}$) in a workflow, where switching agents midway incurs significant re-computation or data transmission costs.

Mechanism Design for Dependent Valuations. Our framework is grounded in Algorithmic Mechanism Design (AMD) (Nisan et al., 2007). While Myerson's optimal auction theory (Myerson, 1981) solves the revenue maximization problem for single items, extending it to combinatorial settings is non-trivial. Prior works have addressed auctions for cloud resources (Zaman and Grosu, 2013) and crowdsourcing (Singla and Krause, 2013), but they generally model resources as additive bundles. The unique challenge in agentic workflows is the *path-dependent switching cost*—a negative externality where the cost of a node depends on the allocation of its prerequisite task (parent node). While general combinatorial auctions with externalities are often NP-hard, we leverage the inherent structured connectivity of agent workflows to design a mechanism that is **computationally tractable in practice**, bridging the gap between theoretical AMD and practical LLM systems.

System Efficiency & Context Caching. Finally, our cost modeling is motivated by physical system constraints. Techniques like *PagedAttention* (Kwon et al., 2023) and *SGLang* (Zheng et al., 2023) optimize memory management, while *Prompt Caching* (Gim et al., 2024) explicitly enables the reuse of KV-caches across requests. Our mechanism is the first to formalize this *system-level optimization* (cache reuse) as an *economic variable* ($S_{u,v}$) within a procurement auction, ensuring that financial incentives align with computational efficiency.

3 System Model

3.1 The Workflow Graph

We represent the agentic workflow as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

- Each node $v \in \mathcal{V}$ represents a sub-task (e.g., "Draft Outline for Chapter 1" or "Generate Code for Step 3").
- Each edge $(u, v) \in \mathcal{E}$ represents a dependency where the input context for node v depends on the output from node u.

For each node u, we assume the workload is intrinsic to the task and fixed across all capable models. Let L_u^{in}, L_u^{out} denote the expected number of input and output tokens generated at node u respectively, and let $L_{u,v}^{trans}$ denote the size of the context passed from u to v. These lengths can be estimated via historical distributions or oracles.

3.2 The Agents and Bids

Let $\mathcal{M} = \{1, \ldots, N\}$ be the set of available LLM agents. We assume feasibility is determined beforehand (e.g., via an oracle like FrugalGPT (Chen et al.)): let $\mathcal{M}_v \subseteq \mathcal{M}$ be the set of agents capable of solving node v. We model the cost structure as follows:

1. **Single Parameter Private Type:** Each agent i has a private base cost c_i per output token. This cost captures the agent's current availability and resource constraints.[1]

2. **Coupled Input-Output Pricing:** For any agent i, the cost of processing an input token is a publicly-known, monotonically increasing function of their per-token output generation cost c_i. For clarity, we adopt a linear model where input cost is a *known fixed fraction* ρ_i of the output cost. This formulation **mirrors the pricing structure** of major commercial APIs (e.g., OpenAI[2], Anthropic[3]), where input tokens are typically priced $2\times$ to $8\times$ lower than output tokens due to the

[1] We assume a *constant marginal cost* model, where unit costs are independent of volume. This is because a single workflow is negligible relative to the service provider's total compute capacity.

[2] https://platform.openai.com/docs/pricing

[3] https://platform.claude.com/docs/en/about-claude/pricing

computational efficiency of parallel prompt processing.

3. **Bidding and Incentives:** Each agent submits a bid b_i, representing their declared price per output token. Crucially, we do not assume that agents are honest when this is not in their interest. Instead, our design objective is to enforce *truthfulness* ($b_i = c_i$) as a *dominant strategy*. This means we construct the payment rules such that a profit-maximizing agent will always achieve the highest utility by reporting their true cost, regardless of the strategies adopted by competitors.

3.3 Context Switching and Communication Costs

Our cost model explicitly captures the overhead of data movement, based on both the Transformer inference architecture and commercial pricing structures:

- **Same-Agent Allocation (Stickiness):** If node u and its child v are assigned to the same agent i (i.e., $A(u) = A(v) = i$), the interaction benefits from *KV-cache reuse* (Kwon et al., 2023). Since the agent retains the attention history in GPU memory, processing the subsequent step does not need to process the upstream information from scratch. This aligns with emerging "prompt caching" pricing models (e.g., Anthropic, DeepSeek) where cached input tokens are charged at a significantly lower rate (often $< 10\%$ of standard input costs).

- **Cross-Agent Allocation (Switching):** If $A(u) \neq A(v)$, a switching cost is incurred due to the loss of shared memory state. The system must pay for agent $A(v)$ to process the upstream information as fresh input (prefill cost). This penalty models the "cold start" processing required when transferring context across disjoint memory domains.

4 STAR: Strategy-proof Topology-aware Agent Routing

We now introduce STAR, a mechanism grounded in the Nobel Prize-winning framework of Myerson (1981) that minimizes the procurer's expected payment while guaranteeing truthful bidding.

4.1 Virtual Cost: The Bridge to Truthfulness

Designing a truthful mechanism poses a unique challenge: we cannot simply select the agents with the lowest raw bids, as this incentivizes agents to game the system (e.g., via strategically inflating their price to marginally undercut competitors). To resolve this, we map the complex strategic game into a standard optimization problem using the concept of *virtual costs*.

Conceptual Intuition. Before we present the formalization, consider the intuition: raw bids b_i are unreliable signals of value. Myerson's theory provides a transformation function $\phi(\cdot)$ that converts a raw bid into a "virtual cost" $\phi_i(b_i)$. This virtual cost can be viewed as the *true economic cost* to the system, encompassing both the agent's physical execution cost and the "incentive cost" required to keep them honest. By running our routing algorithms on these *virtual costs* instead of raw bids, we effectively internalize the strategic externalities. Consequently, the routing algorithm no longer needs to explicitly defend against potential dishonesty; it simply minimizes the sum of virtual costs, and truthfulness follows automatically.

Mathematical Definition. Formally, under the standard assumption that agent costs are drawn from known prior distributions (CDF F_i, PDF f_i), the virtual cost is defined as:

$$\phi_i(b_i) = \underbrace{b_i}_{\text{Raw Bid}} + \underbrace{\frac{F_i(b_i)}{f_i(b_i)}}_{\text{Incentive Markup}} \tag{1}$$

The term $\frac{F_i(b_i)}{f_i(b_i)}$ represents the *information rent*—the profit margin the system implicitly pays to maintain truthfulness. Crucially, optimizing raw bids ignores this cost. By adding this markup, the virtual cost functions as a comprehensive accounting metric: it converts the agent's private resource cost into the procurer's *total payment liability*. Minimizing this metric allows STAR to steer the selection away from agents who might be computationally cheap but require high incentive payments to remain honest.

Ensuring Truthfulness via Monotonicity. Myerson (1981) dictates that a mechanism is truthful if winning probabilities decrease as bids increase (monotonicity). Since the virtual cost function $\phi_i(b_i)$ is monotonically increasing with respect to

the bid b_i,[4] our algorithm naturally favors lower bids. An agent who inflates their bid increases their virtual cost, thereby reducing their likelihood of being selected in the optimal path. Thus, truthfulness is satisfied *by construction*.

Generalizing from Single Agent to Graph Workflow. We build upon Myerson's Lemma, which guarantees that for any single-parameter agent, the expected payment in a truthful mechanism equals their expected virtual cost. We extend this theoretical guarantee to graph-based workflows by leveraging the *linearity of expectation*. Since the total procurement cost is simply the sum of costs across all selected agents, the expected total payment decomposes into the sum of expected payments for each constituent node. Consequently, we can aggregate Myerson's node-level equivalence to the entire graph:

$$\begin{aligned}\mathbb{E}[\text{Total Payment}] &= \sum_{v\in\mathcal{V}} \mathbb{E}[\text{Payment}_v] \\ &= \sum_{v\in\mathcal{V}} \mathbb{E}[\text{Virtual Cost}_v].\end{aligned}$$

This transformation reduces the complex economic game into a tractable algorithmic optimization task: by minimizing the sum of virtual costs $\phi_i(b_i)$, we mathematically guarantee the minimization of the procurer's total expected payment.

4.2 The Optimization Objective

The objective of STAR is to find an allocation mapping $A : \mathcal{V} \to \mathcal{M}$ that minimizes the total virtual cost of the workflow. The objective function consists of two components: the **generation cost** (intrinsic to nodes) and the **switching cost** (dependent on edges and allocation).

We formulate the objective function as follows:

$$\begin{aligned}\min_{A} \quad \mathcal{J}(A) &:= \sum_{v\in\mathcal{V}} G(v, A(v)) \\ &+ \sum_{(u,v)\in\mathcal{E}} \mathbb{I}(A(u) \neq A(v)) \cdot S_{u,v}(A) \quad (2)\end{aligned}$$

The first term captures the *generation cost*, while the second term captures the *switching cost* incurred when adjacent nodes are assigned to different agents. Specifically:

- $G(v, i) = \phi_i(b_i) \cdot L_v^{out}$. This denotes the virtual generation cost for agent i to generate the output for node v.
- $\mathbb{I}(\cdot)$ is the indicator function, which equals 1 if the model switches between parent u and child v, and 0 otherwise (representing the savings from context caching).
- $S_{u,v}(A)$ represents the virtual cost of communication overhead when a switch occurs from agent $A(u)$ to agent $A(v)$. Based on our cost structure, this implies paying for the input processing at the new node:

$$S_{u,v}(A) = \left(\phi_{A(v)}(b_{A(v)}) \cdot \rho_{A(v)}\right) \cdot L_{u,v}^{trans} \quad (3)$$

where $L_{u,v}^{trans}$ is the context transferred to the subsequent node v as input (depending on the system architecture, this term could be exactly L_u^{out}, some state variables, cumulative contexts, or other formats). To maintain generality, we define $S_{u,v}(A)$ as the aggregate virtual cost associated with the context transfer.

4.3 Allocation Algorithms

As established in the previous subsections, to guarantee the incentive compatibility of the mechanism, the allocation rule must strictly satisfy the monotonicity condition. This requires finding the *exact* global minimum of the total virtual cost (Eq. (2)). If the solver is not exact, the approximation solution can lead to the absurd paradox where *"lowering one's bid causes one to lose the allocation,"* thereby fundamentally breaking the mechanism's strategy-proofness. Consequently, we propose a hierarchy of exact algorithms tailored to the graph topology. In all formulations, let $\phi_i(b_{v,i})$ denote the virtual generation cost of agent i for node v.

4.3.1 Algorithm I: Dynamic Programming

The first approach leverages the sparsity of agent interaction graphs. We employ Dynamic Programming (DP) to exploit the graph topology.

Special Case: Trees and Chains. For workflows structured as rooted trees (e.g., hierarchical planning) or linear chains, the optimal allocation can be solved efficiently via *Dynamic Programming (DP)*.

Let $D[v, i]$ denote the minimum total virtual cost for the subtree rooted at v, given that node v is

[4] We assume standard distribution regularity. For irregular distributions, Myerson's *ironing* technique restores monotonicity.

assigned to agent i. The recurrence relation is:

$$D[v,i] = G(v,i) + \sum_{u \in \text{children}(v)} \min_{j \in \mathcal{M}} \Big(D[u,j] + \mathbb{I}(i \neq j) \cdot S_{v,u}(i,j) \Big)$$

where $S_{v,u}(i,j)$ represents the virtual cost of context transfer from agent i to agent j. This approach yields an exact solution in $O(|\mathcal{V}| \cdot |\mathcal{M}|^2)$ time.

General Graphs. Real-world workflows can be more complicated than trees or chains, containing cycles or multi-parent dependencies where standard DP fails. To handle these cases, we employ the *Junction Tree algorithm* based on *tree decomposition* (Cygan et al., 2015).

Intuitively, this method transforms an arbitrary graph with cycles into a tree structure of "clusters" (called bags). By grouping cyclically dependent nodes into the same bag, we can perform dynamic programming over the tree of bags rather than the nodes themselves. This approach guarantees finding the *exact global minimum*. The time complexity is $O(|\mathcal{V}| \cdot |\mathcal{M}|^{w+1})$, where w is the graph's *treewidth*. Since human-designed agent workflows are typically sparse and modular (implying a small w), this algorithm remains highly efficient in practice. We provide the rigorous mathematical definition and the detailed DP transitions for tree decomposition in Appendix B.

4.3.2 Algorithm II: Mixed-Integer Linear Programming (MILP)

Alternatively, we propose another general method, applicable to any workflow topologies. The problem can be formulated as a Mixed-Integer Linear Program (MILP). We leverage the specific cost structure of LLM inference: context transfer costs are determined by the *destination* agent's input token price (re-processing the context), unless the agent is preserved from the previous node, in which case the cost is zero.

Let $L_{u,v}$ denote the size of the context transferred on edge (u,v). Let p_i^{in} and p_i^{gen} be the input and generation virtual prices for agent i. We introduce decision variable $x_{v,i} \in \{0,1\}$ (agent i assigned to node v) and a *cache-miss indicator* $\delta_{u,v,i} \in [0,1]$.

The following MILP minimizes the total virtual cost:

$$\min \sum_{v \in \mathcal{V}, i \in \mathcal{M}} G(v,i) \cdot x_{v,i} + \sum_{(u,v) \in \mathcal{E}, i \in \mathcal{M}} S_{u,v}(i,j) \cdot \delta_{u,v,i}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{M}} x_{v,i} = 1, \quad \forall v \in \mathcal{V}$$

$$\delta_{u,v,i} \geq x_{v,i} - x_{u,i}, \ \forall (u,v) \in \mathcal{E}, \ i \in \mathcal{M}$$

$$x_{v,i} \in \{0,1\}, \quad \delta_{u,v,i} \geq 0$$

The second constraint enforces the KV-Cache logic efficiently. If agent i is preserved ($x_{v,i} = 1, x_{u,i} = 1$), the right-hand side becomes 0, allowing $\delta_{u,v,i} = 0$ (zero cost) via minimization. Modern solvers (e.g., Gurobi) can solve such instances ($|\mathcal{V}| < 100$) to exact optimality in seconds.

4.4 Payment Rule

According to Myerson's optimal auction theory (Myerson, 1981), to guarantee truthfulness, the payment must be determined by a threshold-based rule. Intuitively, for each portion of the assigned workload, the agent is paid the *critical bid value* at which the system would have found it optimal to switch that workload to a competitor. In our workflow setting, an agent i's allocation quantity $q_i(b_i)$ is defined as the *aggregate workload* across their set of assigned nodes $\mathcal{V}_i$. Specifically, the workload for a node v is modeled as $\omega_v = \rho_i \cdot L_v^{in} + L_v^{out}$, accounting for both context processing and generation.

Since $q_i(b_i) = \sum_{v \in \mathcal{V}_i} \omega_v$ is a step-wise decreasing function of the bid, the payment P_i is calculated as the cumulative sum at each "tipping point" where the agent loses a portion of their workload as their bid b_i increases. Formally, let $z_1 < z_2 < \cdots < z_{|\mathcal{V}_i|}$ be the critical bid values (breakpoints) at which $q_i(b_i)$ drops. The total payment P_i is:

$$P_i = \sum_{k=1}^{|\mathcal{V}_i|} z_k \cdot \underbrace{(q_i(z_k - \epsilon) - q_i(z_k + \epsilon))}_{\Delta q_i(z_k)}, \qquad (4)$$

where $\Delta q_i(z_k)$ represents the *workload lost* at price z_k: it is the sum of the workload ω_v for the specific subset of nodes that agent i ceases to win when their bid increases to z_k. To compute this efficiently, we employ a recursive interval-splitting binary search (see Appendix C).

5 Experiments

Table 1 summarizes the performance across experimental trials. STAR consistently outperforms the

baselines, achieving cost reductions of up to 33.6%.

We evaluate STAR against baseline strategies on simulation frameworks designed to mirror real-world LLM marketplace dynamics. Our experiments aim to quantify the cost savings achieved by internalizing topological dependencies and to identify the specific structural patterns where STAR yields the highest cost saving. More details are given in Appendix D.

5.1 Experimental Setup

To ensure our evaluation reflects practical deployment conditions, we instantiate our simulation environment using real-world pricing dynamics derived from commercial LLM markets and topologies representative of standard agentic workflows.

Agent Market Construction (Supply Side). We simulate a marketplace of $N = 9$ agents categorized into three tiers, reflecting the current landscape of commercial LLM providers. Each tier contains 3 distinct agents with base prices centered around the following anchors: **Tier 1 (Small)**, base output price $C^{base} = \$0.10 - \$0.30$ / 1M tokens; **Tier 2 (Medium)**, $C^{base} = \$0.50 - \$1.50$ / 1M tokens; and **Tier 3 (Large)**, $C^{base} = \$2.50 - \$7.50$ / 1M tokens. The price ratio between tiers follows a steep $1 : 5 : 25$ progression, mirroring the cost disparity among tiers of commercial models.[5]

Cost Dynamics (Supply Side). To emulate the *volatility* of LLM model marketplaces, we model the instantaneous generation cost $c_i(t)$ (take agent i as example) as a function of base price, spot scarcity, and electricity cycles. For each trial, we sample a random time $t \sim \text{Uniform}(0, 24)$ and define the cost as:

$$c_i = C_i^{base} \cdot \mathcal{S}_i(\theta_i) \cdot (1 + \beta \cdot \mathcal{E}(t)), \qquad (5)$$

where (1) $\mathcal{E}(t)$ is the diurnal electricity price factor at time t, modeled as a sinusoidal function. Since the time t is *common knowledge* to all participants, the term acts as a global scaling factor; (2) $\mathcal{S}_i \sim \text{Uniform}(0.1, 1.0)$ is the spot scarcity multiplier, representing agent i's specific resource contention (e.g., GPU utilization, local queue depth)[6]. This stochastic factor determines the agent's private cost c_i, which the mechanism must truthfully elicit.

Workflow Topologies (Demand Side). We assume that each node v in the workflow is assigned an intrinsic difficulty level $\text{Diff} \in \{1, 2, 3\}$. An agent i is considered *capable* of handling node v if and only if its capability tier meets or exceeds the task difficulty. Infeasible assignments are strictly prohibited. Based on this, we construct four distinct workflow topologies representing common agentic patterns. Detailed specifications and diagrams are provided in Appendix **??**.

- **S1: Chain of Broadcasters (Creative Writing).** Simulating an agentic workflow solving LongWriter (Bai et al., 2024) tasks. This topology features a sequential chain where a *Brainstorm* node feeds several parallel writers. Crucially, their outputs are collected by an *Aggregator* which then broadcasts a massive compiled context to downstream revisers. This cycle of aggregation and broadcast repeats multiple times. This topology tests the router's ability to manage **context accumulation** and optimize serial bottlenecks where data volume grows progressively.

- **S2: Nested Fan-out (Software Development).** Simulating an agentic workflow solving SWE-Bench (Jimenez et al., 2023) tasks. A root sequence feeds into a *SpecGenerator* (Diff=1) which broadcasts massive specifications to 10 parallel coding modules, which *further* recursively decompose into 30 file-level tasks. This topology features **explosive branching factors**, testing the router's ability to minimize switching costs across deep, high-volume context cascades.

- **S3: Tree-structure Fan-out (Web Development).** Modeling a full-stack agentic team structure for website design (Niu et al., 2025) tasks. An *Architect* designs a system, and a *SpecWriter* broadcasts to 3 **heavy-weight** Team Leads (Diff=2, Output=50k), who then distribute tasks to 12 developers. Unlike S2's sheer scale, this structure emphasizes **intermediate bottlenecks**: the router must optimize the Team Leads' allocation to carefully balance upstream reception costs against downstream distribution costs.

[5] The output price for GPT-5-nano, GPT-5-mini, and GPT-5 API are \$0.4, \$2.0, and \$10.0 per 1M tokens, respectively (which follows $1 : 5 : 25$ progression) as of January 2026.

[6] This range is calibrated to real-world cloud pricing. As of January 2026, Google Cloud Spot VMs provide discounts of up to 91% off on-demand rates during off-peak periods (similar rate for AWS Spot Instances), corresponding to a multiplier lower bound of ≈ 0.1.

Setting	Average Total Payment ($) (Mean ± Std)			Payment Reduction	
	STAR (Ours)	Myopic	Homogeneous	STAR vs. Myopic	STAR vs. Homo.
S1: Chain of Broadcasters	**4.03** ± 0.81	4.48 ± 0.85	5.14 ± 0.98	**10.0%**	**21.6%**
S2: Nested Fan-out	**5.18** ± 0.44	6.72 ± 0.53	7.80 ± 0.64	**22.9%**	**33.6%**
S3: Tree-structure Fan-out	**1.90** ± 0.23	2.10 ± 0.25	2.38 ± 0.29	**9.5%**	**20.2%**
S4: Hourglass-structure	**1.58** ± 0.19	1.96 ± 0.24	2.24 ± 0.28	**19.4%**	**29.5%**

Table 1: **Main experimental results (averaged over 50 trials).** We compare the total procurement payments of STAR against two truthful baselines. STAR consistently minimizes procurement costs compared to truthful baselines across all settings.

- **S4: Hourglass-structure (Legal Analysis).** An iterative map-reduce topology modeling litigation support. It begins with a computationally simple *Case Retriever* fetching massive context, which is consumed by multiple parallel *Analysts*. Crucially, these parallel streams **converge** into a *Lead Attorney* node for synthesis, followed by a **second fan-out stage** where the synthesized strategy is decomposed into parallel drafting tasks. This structure tests the handling of contraction-expansion cycles typical in complex reasoning tasks.

Baselines. We compare STAR against two standard baseline strategies. To ensure a fair comparison of economic efficiency, we implement both baselines as *truthful mechanisms* using second-price payment rules:

- **Myopic Allocation:** Selects the cheapest capable agent for each node independently. The payment for each node is determined by the second-lowest bid among capable agents for that specific task.
- **Homogeneous Allocation:** Auctions the entire workflow as a single indivisible bundle. The contract is awarded to the provider with the lowest aggregate bid who meets the maximum difficulty requirement of the graph. The payment is set to the second-lowest aggregate bid from the pool of capable competitors.

5.2 Results and Analysis

The most significant gains (33.6% in S1, 29.5% in S4) occur in topologies featuring a specific structural motif we term the *"Broadcaster Trap"*. In S4 (Legal), the *Case Retriever* is computationally simple (Diff=1) but outputs 400k tokens to 5 downstream analysts. A myopic router selects a cheap Tier-1 model (∼$0.10) for the retriever. However, this forces the system to pay for transferring 400k tokens to 5 separate analysts, incurring a massive communication penalty. STAR anticipates this and assigns a more expensive Tier-3 model to the retriever if it matches the downstream analysts, eliminating the transfer cost.

The Homogeneous baseline consistently performs the worst. In S1 (SWE), over 60% of the nodes (e.g., file generation, testing) are Diff=1, yet the presence of a single Diff=3 *Planner* node forces the homogeneous strategy to assign the entire graph to an expensive Tier-3 agent. This results in massive resource over-provisioning. STAR successfully identifies "islands of simplicity" within the graph, routing them to cheaper models while selectively enforcing agent stickiness along some high-bandwidth edges..

6 Conclusion

In this work, we addressed the critical challenge of ensuring economic efficiency and incentive compatibility in the emerging landscape of decentralized agentic workflows. We introduced STAR, a mechanism that bridges the gap between algorithmic mechanism design and system-level LLM orchestration. By leveraging virtual cost minimization, STAR simultaneously guarantees that truthful cost reporting is a dominant strategy for self-interested agents and internalizes the path-dependent externalities of context transfer.

Empirically, STAR achieves cost reductions of up to 33.6% over the baselines, effectively mitigating topological inefficiencies like the "broadcaster trap." As the LLM ecosystem transitions to an open marketplace of specialized agents, frameworks like STAR will be essential for orchestrating these resources efficiently, ensuring that the division of labor in AI remains economically viable at scale.

Limitations

While STAR provides a rigorous theoretical framework and demonstrates significant empirical cost savings, we acknowledge several limitations inherent to our modeling assumptions, which point towards fruitful directions for future research.

Single-Dimensional Optimization Objective. Our current mechanism focuses exclusively on minimizing the *financial execution cost* of workflows. While we enforce capability constraints (i.e., ensuring tasks are assigned to capable agents via the tier system), we do not explicitly optimize for other quality-of-service (QoS) metrics such as *end-to-end latency* or *throughput*. In time-sensitive scenarios where latency is as critical as cost, the objective function in Eq. (2) would need to be extended to a multi-objective formulation, potentially requiring a multi-dimensional auction mechanism.

Assumption of Known Priors. Following standard practices in optimal auction design (Myerson, 1981), STAR assumes the orchestrator has access to the prior distributions (F_i) of agents' private costs. In a mature marketplace, these can be estimated from historical bidding data. However, in a *cold-start* scenario with new agents or shifting market dynamics, these priors may be initially inaccurate. Future work could explore *online mechanism design* or bandit-based approaches (one similar setting has been recently discussed by Patra et al. (2026)) to learn these distributions dynamically while maintaining incentive compatibility.

Agent Rationality and Collusion. Our theoretical guarantees rely on the assumption that agents are rational, risk-neutral, and non-collusive. While Strategy-Proofness (DSIC) protects against individual strategic misreporting, it does not account for complex collusive behaviors (e.g., agents forming a cartel to artificially inflate bids). Although standard in mechanism design literature, addressing robust mechanism design against irrational or collusive agents in decentralized AI systems remains an open challenge.

References

Yushi Bai, Jiajie Zhang, Xin Lv, Linzhi Zheng, Siqi Zhu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longwriter: Unleashing 10,000+ word generation from long context llms. *arXiv preprint arXiv:2408.07055*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.

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

Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. 2015. *Parameterized algorithms*, volume 5. Springer.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Roger B Myerson. 1981. Optimal auction design. *Mathematics of operations research*, 6(1):58–73.

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani, editors. 2007. *Algorithmic Game Theory*. Cambridge University Press.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. 2025. Flow: Modularized agentic workflow automation. In *International Conference on Representation Learning*, volume 2025, pages 74949–74977.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

Pronoy Patra, Sankarshan Damle, Manisha Padala, and Sujit Gujar. 2026. Truthful reverse auctions for adaptive selection via contextual multi-armed bandits. In *The 25th International Conference on Autonomous Agents and Multi-Agent Systems*.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186.

Adish Singla and Andreas Krause. 2013. Truthful incentives in crowdsourcing tasks using regret minimization mechanisms. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1167–1178.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Sharrukh Zaman and Daniel Grosu. 2013. A combinatorial auction-based mechanism for dynamic vm provisioning and allocation in clouds. *IEEE Transactions on Cloud Computing*, 1(2):129–141.

Yi-Kai Zhang, Shiyin Lu, Qing-Guo Chen, Weihua Luo, De-Chuan Zhan, and Han-Jia Ye. 2025. Let the llm stick to its strengths: Learning to route economical llm. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody_Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2023. Efficiently programming large language models using sglang.

A Derivation of Virtual Cost for Procurement

In this section, we provide the formal derivation showing that minimizing the expected total payment in a procurement (reverse) auction is equivalent to minimizing the expected virtual cost. This derivation adapts Myerson's optimal auction theory (Myerson, 1981) from the standard revenue maximization setting to our cost minimization setting.

Setup. Consider a single agent i with private cost c_i drawn independently from a distribution with CDF F_i and PDF f_i on support $[\underline{c}, \bar{c}]$. The mechanism specifies:

- Allocation rule $x_i(c_i)$: The probability (or amount) of task assignment given reported cost c_i.
- Payment rule $p_i(c_i)$: The payment from the procurer to the agent.

The agent's utility is $u_i(c_i) = p_i(c_i) - c_i x_i(c_i)$.

Incentive Compatibility (IC). By the Envelope Theorem, for a mechanism to be truthful, the utility function must satisfy $u_i'(c_i) = -x_i(c_i)$. This implies that the utility of an agent with cost c_i is determined by the allocation rule:

$$u_i(c_i) = u_i(\bar{c}) + \int_{c_i}^{\bar{c}} x_i(z)\, dz \tag{6}$$

We assume individual rationality such that the highest-cost agent makes zero profit, i.e., $u_i(\bar{c}) = 0$. Thus, $u_i(c_i) = \int_{c_i}^{\bar{c}} x_i(z)\, dz$.

Expected Payment. The procurer's expected payment to agent i is:

$$\mathbb{E}[p_i] = \int_{\underline{c}}^{\bar{c}} p_i(c) f_i(c)\, dc \tag{7}$$

Substituting $p_i(c) = u_i(c) + c x_i(c)$:

$$\mathbb{E}[p_i] = \int_{\underline{c}}^{\bar{c}} \left(\int_{c}^{\bar{c}} x_i(z)\, dz + c x_i(c) \right) f_i(c)\, dc \tag{8}$$

Integration by Parts. We simplify the double integral term using integration by parts. Let $U = \int_c^{\bar{c}} x_i(z)\, dz$ and $dV = f_i(c)dc$. Then $dU = -x_i(c)dc$ and $V = F_i(c)$.

$$\int_{\underline{c}}^{\bar{c}} \left(\int_{c}^{\bar{c}} x_i(z)\, dz \right) f_i(c)\, dc \tag{9}$$

$$= \left[F_i(c) \int_{c}^{\bar{c}} x_i(z)\, dz \right]_{\underline{c}}^{\bar{c}} - \int_{\underline{c}}^{\bar{c}} F_i(c)(-x_i(c))\, dc \tag{10}$$

$$= 0 - 0 + \int_{\underline{c}}^{\bar{c}} F_i(c) x_i(c)\, dc \tag{11}$$

$$= \int_{\underline{c}}^{\bar{c}} \frac{F_i(c)}{f_i(c)} x_i(c) f_i(c)\, dc \tag{12}$$

(Note: The boundary terms vanish because at $\bar{c}$, the integral is 0; at $\underline{c}$, $F_i(\underline{c}) = 0$.)

Result. Substituting this back into the expected payment equation:

$$\mathbb{E}[p_i] \tag{13}$$

$$= \int_{\underline{c}}^{\bar{c}} \left(\frac{F_i(c)}{f_i(c)} x_i(c) + c x_i(c) \right) f_i(c)\, dc \tag{14}$$

$$= \int_{\underline{c}}^{\bar{c}} \underbrace{\left(c + \frac{F_i(c)}{f_i(c)} \right)}_{\phi_i(c)} x_i(c) f_i(c)\, dc \tag{15}$$

$$= \mathbb{E}_{c_i}[\phi_i(c_i) x_i(c_i)] \tag{16}$$

Thus, minimizing the expected payment is mathematically equivalent to minimizing the expected virtual cost $\phi_i(c) = c + \frac{F_i(c)}{f_i(c)}$.

B Tree Decomposition for General Graphs

For general graphs containing cycles, we utilize the Junction Tree algorithm. This involves two steps: constructing a tree decomposition and running dynamic programming on the resulting structure.

1. Decomposition Definition. A tree decomposition of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a tree $\mathcal{T}$ where each node represents a "bag" $B_t \subseteq \mathcal{V}$. This decomposition must satisfy two properties:

1. **Edge Coverage:** For every edge $(u, v) \in \mathcal{E}$, there exists at least one bag B_t containing both u and v.
2. **Running Intersection:** For any vertex $v \in \mathcal{V}$, the set of bags containing v forms a connected subtree in $\mathcal{T}$.

The *width* of the decomposition is $\max_t |B_t| - 1$. The *treewidth* w of $\mathcal{G}$ is the minimum width over all possible decompositions.

2. Dynamic Programming Transition. We treat the junction tree $\mathcal{T}$ as a rooted tree and perform bottom-up DP. Let $\mathbf{a}_t \in \mathcal{M}^{|B_t|}$ represent a joint assignment of agents to all nodes within bag B_t. The state space size for each bag is $|\mathcal{M}|^{|B_t|}$. Let $D_t(\mathbf{a}_t)$ denote the minimum virtual cost for the subtree of bags rooted at B_t, consistent with the local assignment $\mathbf{a}_t$. The recurrence relation is:

$$D_t(\mathbf{a}_t) = \Psi(B_t, \mathbf{a}_t) + \sum_{B_c \in \text{children}(B_t)} \min_{\substack{\mathbf{a}_c \in \mathcal{M}^{|B_c|} \\ \mathbf{a}_c \sim \mathbf{a}_t}} \left(D_c(\mathbf{a}_c) - \Delta(B_t, B_c) \right)$$

Here, the terms are defined as follows:

- $\Psi(B_t, \mathbf{a}_t)$: The local cost within bag B_t, summing the generation costs of nodes in B_t and the switching costs for edges fully contained in B_t.
- $\mathbf{a}_c \sim \mathbf{a}_t$: The consistency constraint. The child assignment $\mathbf{a}_c$ must match the parent assignment $\mathbf{a}_t$ for all nodes in the separator (intersection) $B_t \cap B_c$.
- $\Delta(B_t, B_c)$: A correction term to prevent double-counting. Since nodes in the intersection $B_t \cap B_c$ appear in both bags, their generation costs are subtracted:

$$\Delta(B_t, B_c) = \sum_{v \in B_t \cap B_c} \phi_{\mathbf{a}_t[v]} \cdot L_v \tag{17}$$

The computational bottleneck is determining the optimal assignment for each bag, scaling as $O(|\mathcal{M}|^{w+1})$. For sparse workflow graphs where w is small (typically $w \leq 3$, e.g., $w = 1$ for linear chains or trees, $w = 2$ for feedback loops or ensemble voting/Best-of-N structure, versus high w for dense cliques), this is computationally tractable.

C Payment Calculation Algorithm

To accurately compute the cumulative payment defined in Eq. (4), we must identify all critical bid values where the allocation quantity $q(b)$ changes. Since $q(b)$ is a step function with potentially multiple drops, a standard binary search (which finds a single root) is insufficient.

We implement a *recursive interval-splitting binary search* algorithm (Algorithm 1). The core idea is to maintain a set of "active intervals" $[L, R]$ where the allocation at the endpoints differs (i.e., $q(L) \neq q(R)$), implying at least one drop exists within. We iteratively split these intervals at their midpoint M, discarding sub-intervals that are constant (flat) and keeping those that contain transitions.

This approach is highly efficient for our setting. If there are K critical drops, the algorithm converges in $\mathcal{O}(K \log(\frac{b_{max}}{\epsilon}))$, focusing computational effort solely on the transition regions.

D Market Simulation Details

D.1 Agent Pricing and Tiers

We model a 9-agent marketplace with a steep price progression $(1 : 5 : 25)$ to simulate the difference

Algorithm 1: Recursive Interval Splitting for Payment Calculation

Require: Winning bid b_{win}, Max bid b_{max}, Allocation function $q(\cdot)$.
Initialize: $Payment \leftarrow 0$.
Initialize Queue: $Q \leftarrow \{[b_{win}, b_{max}]\}$.
while Q is not empty **do**
  Pop interval $[L, R]$ from Q.
  Calculate midpoint $M \leftarrow (L + R)/2$.
  Evaluate allocations: $q_L, q_M, q_R \leftarrow q(L), q(M), q(R)$.
  if $q_L == q_R$ **then**
    continue ▷ Interval is flat, no price drop here.
  end if
  if $|L - R| < \epsilon$ **then**
    ▷ Found a critical point at approx M
    $Drop \leftarrow q_L - q_R$
    $Payment \leftarrow Payment + M \cdot Drop$
    continue
  end if
  ▷ Split and check sub-intervals
  if $q_L \neq q_M$ **then**
    Push $[L, M]$ to Q.
  end if
  if $q_M \neq q_R$ **then**
    Push $[M, R]$ to Q.
  end if
end while
return $Payment$

between distilled models and frontier reasoning models. The specific base prices for the agents are fixed as follows (before dynamic multipliers):

Tier	Base Prices ($/1M)	Avg Price
Tier 1 (Small)	$0.10, $0.20, $0.30	$0.20
Tier 2 (Med)	$0.50, $1.00, $1.50	$1.00
Tier 3 (Large)	$2.50, $5.00, $7.50	$5.00

Table 2: Base pricing configuration. Within each tier, agents have 3x variance to encourage competition.

D.2 Dynamic Cost Model

To simulate real-time market conditions, the actual cost $C_i(t)$ is dynamically generated in each trial using the following components:

1. **Spot Multiplier** $\mathcal{S}_i(t)$**:** Drawn from Uniform$(0.1, 1.0)$. This introduces a 10x dynamic range, simulating periods of extreme surplus (90% off). Set $\beta = 1.5$.

2. **Electricity Factor** $\mathcal{E}(t)$**:** A sinusoidal function $\mathcal{E}(t) = 0.5 \cdot (1 + \sin(\frac{2\pi(t-8)}{24}))$ simulating grid load, where $t \in [0, 24]$ is the trial start time.

To test robustness, token lengths for each node are perturbed by $\pm 10\%$ noise in each trial.

D.3 S1: Chain of Broadcasters (Creative Writing)

Simulating a long-context creative writing pipeline (e.g., LongWriter), this topology features a **sequential bottleneck** architecture. Unlike tree-like expansions, S1 operates in a rhythm of expansion, aggregation, and re-broadcast.

The workflow begins with parallel content generation by multiple *Chapter Writers*. These outputs are collected by a central *Aggregator*, which synthesizes the full narrative into a 300k-token context. This massive context is then broadcast to a second parallel layer of *Revisers*. The *Aggregator* creates a high-stakes "Broadcaster Trap": assigning it to a cheap but isolated provider incurs heavy transfer penalties for all downstream Revisers. The topology structure is visualized in Figure 1, and the configuration details are listed in Table 3.

D.4 S2: Nested Fan-out (Software Engineering)

This topology mimics a complex software engineering workflow derived from SWE-Bench tasks. It is characterized by a **deep, recursive decomposition** structure. The workflow begins with a sequential analysis phase (Scanner, Analyzer, Planner) which culminates in a *SpecGenerator*.

The *SpecGenerator* acts as the critical broadcaster, generating a massive 350k-token technical specification. This context is broadcast to a primary fan-out layer of 10 distinct *Modules*, which subsequently branch further into a secondary layer of 30 granular *File-level* tasks. The workflow concludes with a convergence phase via Merger and Tester nodes. This structure tests the mechanism's ability to handle multi-stage switching costs where a decision at the root propagates through two layers of descendants. The topology structure is visualized in Figure 2, and the detailed node configuration is provided in Table 4.

Table 3: **S1 Node Configuration.** Highlights the context accumulation at the *Aggregator*, creating a high-cost switching trap for downstream revisers.

Node Role	**Count**	**Diff** (D_v)	**Input** (L_{in})	**Output** (L_{out})
Brainstorm	1	3	2k	30k
Parallel Writing				
Chapter Writers	8	3	30k	40k
Aggregator	1	1	320k	300k
Parallel Revising				
Reviser (Polish)	4	1	300k	35k
Reviser (Content)	4	3	300k	35k
Final Editor	1	3	35k	10k

Table 4: **S2 Node Configuration.** The *SpecGenerator* acts as a massive broadcaster (350k tokens) feeding into a recursive fan-out structure.

Node Role	**Count**	**Diff** (D_v)	**Input** (L_{in})	**Output** (L_{out})
Scanner	1	1	1k	3k
Analyzer	1	2	3k	5k
Planner	1	3	5k	8k
SpecGenerator	1	1	8k	350k
Fan-out Layer 1 (Modules)				
Module (Core Logic)	1	3	350k	25k
Module (API/Data/UI)	7	2	350k	25k
Module (Utils/Cache)	2	1	350k	25k
Fan-out Layer 2 (Files)				
File (Main Implementation)	10	3	25k	15k
File (Types/Boilerplate)	20	1	25k	15k
Merger	1	1	30k	15k
Tester	1	2	15k	8k

D.5 S3: Tree-structure Fan-out (Web Development)

Modeled after a full-stack web development team, S3 employs a shallow but wide **Tree-structure Fan-out**. This topology is distinct from S2 due to the presence of **heavy intermediate nodes**.

A *SpecWriter* broadcasts a 300k-token requirement document not to leaf workers, but to three *Team Leads* (Frontend, Backend, Interactive). Crucially, these Team Leads are complex nodes (Diff=2) with substantial output generation (40k-50k tokens), making them expensive to migrate. They subsequently distribute tasks to a leaf layer of 12 Developers. This structure tests the mechanism's ability to balance costs across a "heavy" middle layer. The topology structure is visualized in Figure 3, with configuration parameters in Table 5.

D.6 S4: Hourglass (Legal Discovery)

The **Hourglass** topology simulates a complex legal discovery process involving map-reduce operations. It features a unique "Retrieval Trap" and a **dual fan-out/fan-in structure**.

A computationally cheap *Case Retriever* (Diff=1) broadcasts a massive 400k-token dataset to a wide layer of Analysts. This is followed by a contraction phase at the *Aggregator*, and a secondary expansion to *Strategist* nodes. This structure creates complex path dependencies, where the initial routing decision at the Retriever significantly impacts the cumulative transfer costs incurred by both the Analyst and Strategist layers. The topology structure is visualized in Figure 4, and the node configuration is detailed in Table 6.

Table 5: **S3 Node Configuration.** The *SpecWriter* broadcasts to three Team Leads. Note that Team Leads are Diff=2 nodes with substantial output (40-50k), making them expensive to switch.

Node Role	**Count**	**Diff** (D_v)	**Input** (L_{in})	**Output** (L_{out})
Architect	1	2	2k	10k
SpecWriter	1	1	10k	300k
Intermediate Layer (Leads)				
Lead (Frontend)	1	2	300k	**50k**
Lead (Backend)	1	2	300k	40k
Lead (Interactive)	1	2	300k	30k
Leaf Layer (Developers)				
Devs (Frontend Static)	5	1	50k	15k
Devs (Backend API)	4	2	40k	20k
Devs (Interactive AI)	3	3	30k	15k

Table 6: **S4 Node Configuration.** The *Case Retriever* (Diff=1) creates a trap: using a small model saves generation cost but incurs massive transfer penalties to the 5 downstream Analysts.

Node Role	**Count**	**Diff** (D_v)	**Input** (L_{in})	**Output** (L_{out})
Expert Assessment	1	3	2k	5k
Case Retriever	1	1	5k	400k
Fan-out Layer 1 (Analysis)				
Analyst (Clause Extract)	3	1	400k	35k
Analyst (Liability)	2	2	400k	20k
Contraction Phase				
Aggregator	1	1	145k	80k
Fan-out Layer 2 (Strategy)				
Strategist (Aggressive)	1	2	80k	20k
Strategist (Defensive)	1	2	80k	20k
Strategist (Settlement)	1	2	80k	20k
Expert Synthesis	1	3	60k	10k

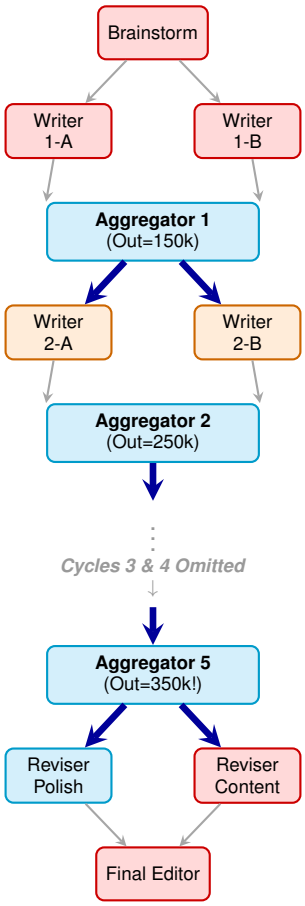

Figure 1: **S1: Chain of Broadcasters (Creative Writing).** A 5-stage iterative pipeline. The diagram visualizes Cycles 1, 2, and 5 (omitting 3 and 4). While each cycles may broadcast to arbitrary numbers of nodes, this diagram illustrates the structural topology. Context accumulates at each simple Aggregator (Diff=1), creating a series of intensifying "Broadcaster Traps" (represented by thick blue arrows) before reaching the final editor.

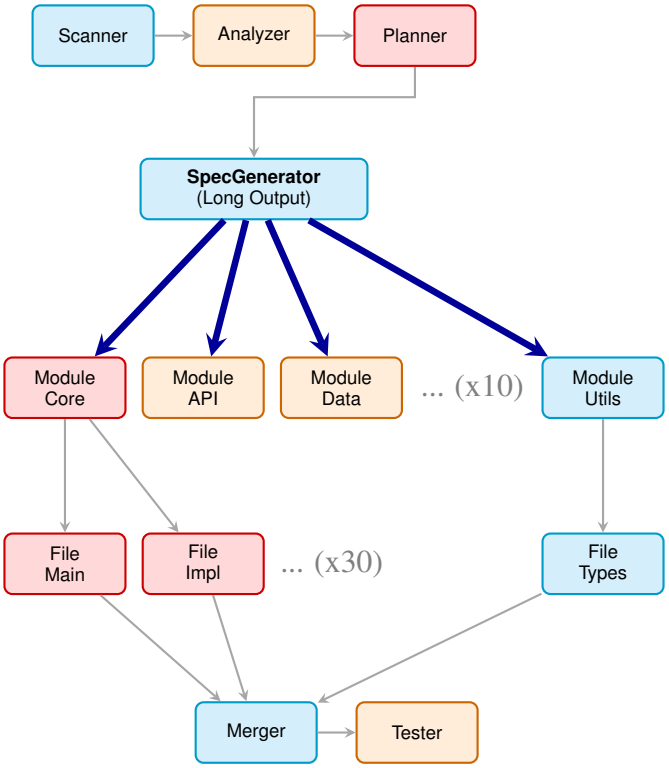

Figure 2: **S2: Nested Fan-out (Software Development).** Features a deep, recursive structure. A simple SpecGenerator (Diff=1) broadcasts massive 350k-token specifications to 10 modules, which further branch into 30 file-level tasks. The thick blue arrows highlight the critical high-bandwidth edges that create cascading switching penalties.

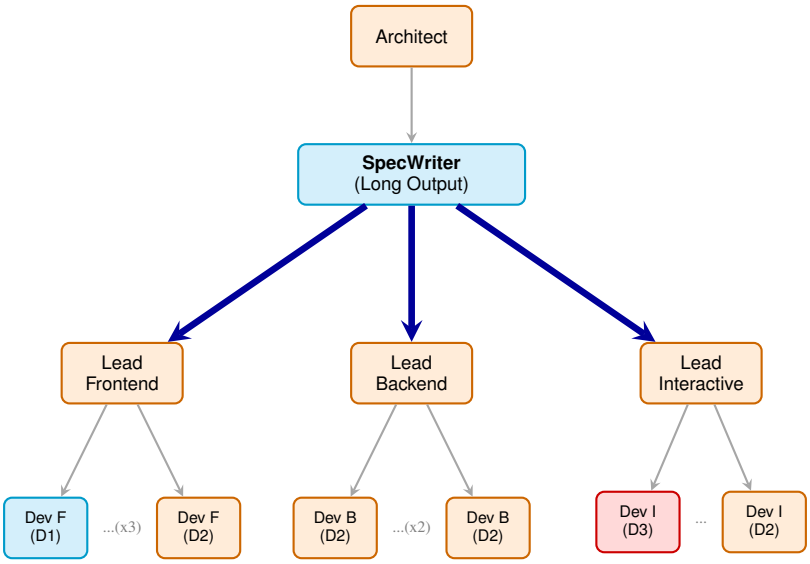

Figure 3: **S3: Tree-structure Fan-out (Web Development).** Emphasizes a single, massive broadcast stage followed by a structured team fan-out. A simple SpecWriter (Diff=1) sends a 250k-token specification to three heavy-weight Team Leads (Diff=2), who then distribute tasks to multiple developers of varying difficulties.

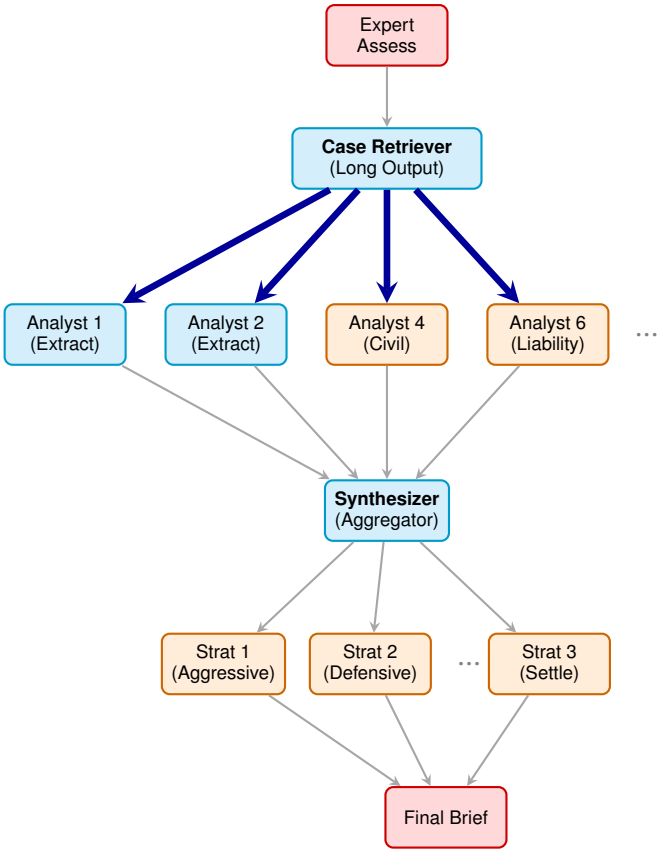

Figure 4: **S4: Hourglass-structure (Legal Analysis).** Visualizing the "Retrieval Trap." A simple Case Retriever (Diff=1) creates a massive bottleneck by broadcasting 400k tokens to downstream analysts (mixed Diff=1/2). The workflow narrows at the simple Synthesizer (Diff=1) before expanding again to Strategists (Diff=2), creating complex path dependencies and multi-stage optimization challenges.