HACO: Hedged Agent Computing for Reliable LLM Systems

Enhan Li[1], **Hongyang Du**[1*]

[1]The University of Hong Kong
lienhan_eee@connect.hku.hk, duhy@hku.hk

Abstract

As large language model (LLM) agents move from isolated prompting to long-horizon workflows, failures increasingly arise at the role-to-instance binding boundary, where task-specific role requests must be assigned to concrete agent instances under current service, network, and query conditions. Existing agent system research has improved role specialization, workflow topology, memory, and tool use, but often assumes a fixed stable execution environment. This assumption limits deployed reliability, because the same role request can exhibit different latency, failure probability, and output quality across agent instances operating under different service regions and network conditions. We propose Hedged Agent Computing (HACO), a runtime control scheme that treats each role request as a reliability-constrained selection problem over candidate agent instances, each coupling a role type, an LLM, and a concrete execution environment. Different from routing, HACO adaptively selects a hedge set of candidates for each invocation. Its allocation rule combines optimistic ranking, which prioritizes candidates with high estimated quality, reliability, and informative uncertainty, with conservative reliability accumulation, which selects candidates toward a target success probability. Through experience harvesting, HACO updates candidate and link profiles from all executed candidate traces, including quality, success, latency, and network statistics. Experiments on various benchmarks, together with runtime degradation studies, show that HACO improves robustness and output quality under changing deployment conditions, while using lower token and latency cost than exhaustive parallel execution.

Code is available at `https://github.com/NICE-HKU/haco.git`.

1 Introduction

LLM-based agent systems, especially multi-agent systems (MAS), have become a common paradigm for complex task execution and advanced automation [24, 41, 17, 51]. Compared with single-LLM execution, agent systems can distribute a task across specialized role types, coordinate intermediate outputs, cross-check partial results, and repair errors through iterative feedback. These coordination gains are especially valuable for tasks that require goal decomposition, intermediate coordination, iterative refinement, and long-horizon execution. As a result, agent systems have become an important direction for building practical AI systems, especially in domains such as data analysis, decision support, and scientific discovery.

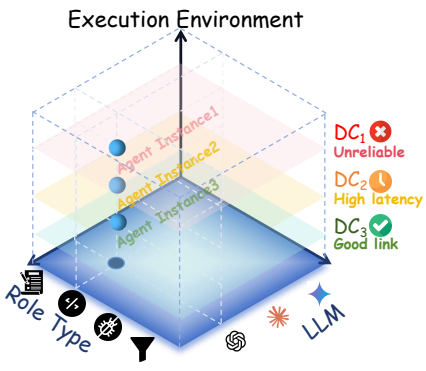


Figure 1: Three coupled axes of role invocation in MAS.

*Corresponding author.

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

This shift has also changed how practical agents are designed. Modern agent systems are increasingly built with complete agent harnesses instead of standalone models with prompts [4, 45]. In practice, reliable execution depends on surrounding components such as tool interfaces, memory management, safety controls, and execution orchestration. Recent work therefore places growing emphasis on persistent state [57], controller design [33], execution layers, and tool coordination [42], because these components often determine whether agents can maintain progress through long and error-prone workflows. However, this system perspective remains primarily organized around software-level MAS design, including role type specialization and system topology of agents. When validating these designs, they often vary LLM backbones in the agent instances to show performance differences and robustness under the same MAS design [54]. As Fig. 1 illustrates, once the system topology is fixed, this view attributes uncertainty mainly to the agent role type or the LLM's capability, such as reasoning or tool-use quality, while treating the execution environment of the agent instance as a fixed background condition. The coupling among the capability of the selected LLM, its role type and the execution environment receives comparatively little attention.

However, the fixed-environment assumption breaks down in real deployments. Even when the MAS software topology and role invocation remain unchanged, the final behavior depends heavily on the agent instance that performs the concrete operation. Specifically, a role invocation may pass through different service providers and execution regions. Its behavior is shaped by both LLM capability and physical environment factors, e.g., network latency, bandwidth fluctuation, transient failures, resource contention, and service interruptions, of the selected agent instance. Consequently, two invocations that are identical at the software level may exhibit substantially different latency, execution reliability, and outcomes under different execution environments. In practice, the role type is typically determined by the MAS design, while LLM choice and runtime conditions remain operational variables. For example, modern LLM serving platforms expose region and deployment choices as runtime or configuration variables, so the same role invocation request may not always execute in the same physical location [34, 18, 1, 37] (More examples are given in Appendix A). Moreover, recent infrastructure incidents, including Red Sea submarine-cable disruptions and Azure's West Europe thermal event, show that physical and regional execution conditions can directly affect service latency and availability [12, 11, 35]. Therefore, agent systems should not attribute uncertainty only to model reasoning or software logic, but should also account for execution location, network paths, and region-level infrastructure state.

We therefore view role invocation as a coupled runtime decision over role type, LLM choice, and execution environment, and extend the conventional agent-instance abstraction by adding execution environment as a third axis. Inspired by the success of redundancy and fault-tolerance mechanisms in distributed systems, we use hedged concurrent execution to manage invocation-level uncertainty in computing and communication environments. HACO focuses on the role-invocation allocation point, which is different from single-route selection in *LLM routing* [13, 36], post-hoc escalation in *fallback or cascade* methods [55, 21], serving-batch construction in *batch inference* [30, 58], and output selection or synthesis in *MoA-style* or *Best-of-N aggregation* [27, 48, 49]. It asks a preceding execution-control question: before a role request is executed, whether multiple candidate agent instances should execute the same request, which candidates should be activated, and when the allocated redundancy is sufficient. Appendix B provides a structured comparison with related runtime and inference-time paradigms (Table 3 and Fig. 7). Motivated by these insights, we propose **Hedged Agent COmputing (HACO)**, where the core selection unit is a *candidate agent instance*, i.e., a concrete role-specific LLM deployment in an execution environment; we use *candidate* as shorthand thereafter. HACO allocates a small hedge set of candidates and uses the resulting execution traces to refine future allocation decisions. Our contributions are summarized as:

- We model role invocation in LLM-based agent systems as a three-axis runtime decision over role type, LLM choice, and execution environment. This formulation exposes the role-to-instance binding boundary as a source of reliability, latency, and quality variation under concrete runtime conditions.

- We introduce HACO, a redundancy-based runtime control paradigm that selects reliability-constrained hedge sets over heterogeneous agent instances. Through experience harvesting, HACO refines candidate and link profiles, allowing the system to adapt redundancy to runtime conditions and balance robustness against execution cost.

- We evaluate HACO across representative benchmarks and stress settings, showing that it improves robustness and output quality while offering a controllable trade-off among system reliability, task success, and execution cost measured by token usage and latency.

2 Related Work and Theory Motivation

Agent harnesses. Recent work increasingly treats LLM agents as complete execution systems supported by agent harnesses, where the surrounding layer manages context construction, memory access, tool invocation, validation, and lifecycle control [4, 45, 57, 33, 42]. This harness view shows that practical agent performance depends on system mechanisms that maintain progress across long-horizon and failure-prone workflows, beyond the capability of the underlying model in a single turn. HACO builds on this system-level perspective, but studies a different control problem, i.e., how to allocate redundant execution when runtime conditions are uncertain.

LLM-based multi-agent systems. Research on LLM-based MAS mainly focuses on how agents are organized, how they communicate, and how they are controlled during complex tasks [22, 51, 43]. Representative systems include workflow-based role specialization such as MetaGPT [24], communication-centric collaboration such as ChatDev [41], and orchestrator-led delegation such as Magentic-One [17]. These works improve collaboration structure, communication protocols, and controller design. HACO addresses a complementary question, i.e., given a fixed MAS workflow, how should each role invocation be executed when candidate capability, network condition, and execution reliability vary at runtime.

Redundant and parallel agent execution. Redundancy is a common mechanism for improving reliability in deployed systems, but it also introduces execution cost [8]. Existing agent methods mainly use parallelism for full generation with post-hoc selection or structured wide search. For example, MoA-style method executes all candidates and selects the final output after generation [48], while A-MapReduce and AgentSwing use structured parallelism for wide search and long-horizon branch routing [7, 14]. Agent-diversity studies further show that parallel agents can provide complementary information [52]. These methods demonstrate the value of parallel execution, but do not decide how much redundancy each role invocation needs under changing runtime conditions. Redundancy is theoretically supported as a stabilizing mechanism under uncertainty and can reduce failure risk in multi-agent execution [5], but excessive redundancy increases token usage, latency, and coordination cost. HACO builds on this principle by treating redundancy as a query-dependent runtime control variable and formulating allocation as a reliability-constrained subset-selection problem over the candidate pool $\mathcal{A}$:

$$\min_{\mathcal{S}\subseteq\mathcal{A}} \sum_{a_i\in\mathcal{S}} C_i \quad \text{s.t.} \quad \mathbb{P}(\text{qualified success} \mid \mathcal{S}) \geq \tau. \tag{1}$$

Here, $\mathcal{S}$ is the hedge set, τ is the target reliability, and C_i penalizes the candidate activation footprint, not realized wall-clock latency; the realized latency is T_{obs} in Section 3.3. Qualified success means that at least one selected candidate executes successfully and meets the quality criterion.

3 The HACO Paradigm

HACO treats redundancy as a runtime control variable, deciding how many and which candidate agent instances to activate for each role invocation. As shown in Fig. 2, HACO consists of four stages, i.e., role abstraction, redundancy allocation, hedged execution, and experience harvesting.

3.1 Phase 1: Role Abstraction

HACO starts from role invocation events produced by the underlying MAS. A role type refers to a functional component such as planner, coder, debugger, or filter. At step t, a role invocation event is

$$e_t = (r_t, x_t), \tag{2}$$

where r_t is the role type to invoke and x_t is the input payload that is constructed from the original query q and the current task context $\mathcal{C}_t$, including intermediate plans, execution results, error messages, and outputs from previous roles. For example, a *debugger* payload may contain the original query together with the failed *coder* output, traceback, and current notebook state.

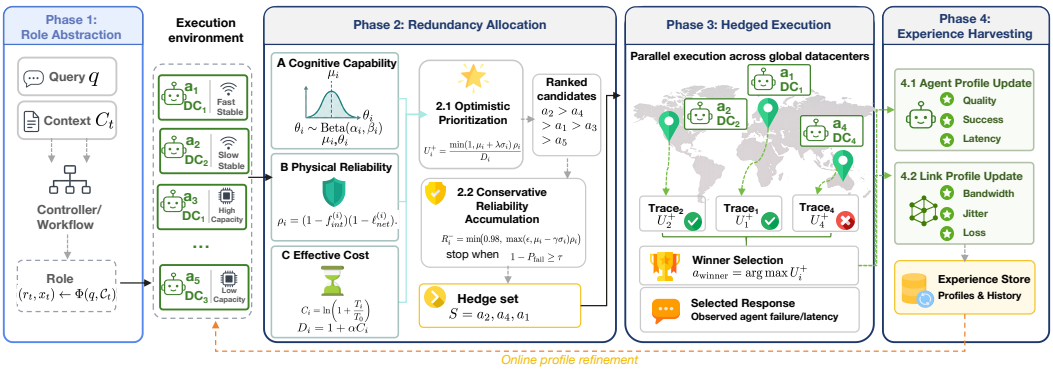


Figure 2: **Overview of the HACO paradigm.**

This abstraction connects the MAS controller to HACO's redundancy allocation mechanism. Given Eq. (2), HACO uses r_t to identify the candidate pool and sends the same payload x_t to the selected candidate agent instances. The controller could be a predefined workflow, a finite-state machine, a decentralized negotiation protocol, a MetaGPT-style workflow [24], or a single-agent controller, depending on MAS design. Without loss of generality, our experiments instantiate HACO with a relatively complex FSM-based controller derived from DATAWISE, as detailed in Appendix C.1. HACO itself only requires the controller to emit role invocation events in the form of Eq. (2).

3.2 Phase 2: Redundancy Allocation

Given a role invocation, HACO selects a hedge set $\mathcal{S} \subseteq \mathcal{A}$ that meets a target reliability level with low latency-aware resource cost. To this end, HACO characterizes each candidate with uncertainty-aware utility components and allocates redundancy through dual-bound coordination.

Candidate Characterization. We characterize each candidate a_i along three dimensions, i.e., uncertain cognitive capability, physical execution reliability, and latency-aware resource cost.

1. Cognitive Capability. Let $\theta_i \in [0, 1]$ denote the expected normalized output-quality capability of candidate a_i. HACO models θ_i using a Beta prior, a standard choice for variables restricted to the unit interval [15, 29, 32]:

$$\theta_i \sim \mathrm{Beta}(\alpha_i, \beta_i), \tag{3}$$

which is initialized with $\alpha_i = \beta_i = 1$. After observing a normalized output-quality score $q_i \in [0, 1]$, HACO applies a lightweight soft-count update:

$$\alpha_i \leftarrow \alpha_i + q_i, \qquad \beta_i \leftarrow \beta_i + (1 - q_i). \tag{4}$$

The posterior mean and standard deviation are

$$\mu_i = \frac{\alpha_i}{\alpha_i + \beta_i}, \qquad \sigma_i = \sqrt{\frac{\alpha_i \beta_i}{(\alpha_i + \beta_i)^2(\alpha_i + \beta_i + 1)}}. \tag{5}$$

Here, μ_i estimates the expected capability of the selected candidate a_i, while σ_i captures uncertainty.

2. Physical Execution Reliability. Let $f_{\mathrm{int}}^{(i)}$ denote the internal failure rate of candidate a_i, and let $\ell_{\mathrm{net}}^{(i)}$ denote the network loss rate. We define the execution success probability as

$$\rho_i = (1 - f_{\mathrm{int}}^{(i)})(1 - \ell_{\mathrm{net}}^{(i)}). \tag{6}$$

Equation (6) treats internal and network failures for a candidate as independent; independence across candidates is a separate assumption in the set-level certificate (Appendix D.2).

3. Latency-Aware Resource Cost. For each candidate, HACO estimates an effective latency by combining the candidate's processing time and the expected communication overhead:

$$T_i = t_{\mathrm{proc}}^{(i)} + \frac{k_i}{\max(\epsilon_{\mathrm{B}}, b_i - \kappa j_i)}. \tag{7}$$

Here, $t_{\text{proc}}^{(i)}$ denotes the historical average processing latency of candidate i, k_i is the estimated message size, b_i is the observed effective bandwidth, j_i is the unit bandwidth jitter, and $\epsilon_{\text{B}} > 0$ is a bandwidth floor. The coefficient $\kappa > 0$ models how strongly bandwidth jitter reduces the effective bandwidth term.

To reflect the diminishing sensitivity of user-perceived delay, HACO defines a scale-normalized logarithmic latency cost with the estimated latency T_i based on the Weber–Fechner law [44]:

$$C_i = \ln\left(1 + \frac{T_i}{T_0}\right), \tag{8}$$

where T_0 is a reference latency, instantiated as the median estimated latency among candidates for the current routing step. HACO converts this latency-aware resource cost into a stable discount factor:

$$D_i = 1 + \eta C_i, \tag{9}$$

where η controls the strength of latency penalization. Compared with directly using C_i, this form preserves the logarithmic delay effect, penalizes larger latency monotonically, and avoids excessive punishment of high-latency candidates and excessive reward for near-zero latency.

Dual-Bound Coordination. Based on the candidate characterization results, HACO allocates redundancy with a dual-bound coordination mechanism. Candidate ranking and redundancy termination face different uncertainty risks. For ranking, overly pessimistic estimates may hide under-observed but potentially strong candidates. For termination, overly optimistic estimates may stop the hedge construction too early and weaken system reliability. HACO addresses these two risks with separate estimates, i.e., an optimistic utility for ranking candidates and a conservative reliability estimate for deciding when the hedge set is sufficient.

1. Optimistic Prioritization. For each candidate a_i, HACO computes an optimism-adjusted utility

$$U_i^{+} = \frac{\min(1, \mu_i + \lambda\sigma_i)\,\rho_i}{D_i}, \tag{10}$$

where μ_i and σ_i are the mean and standard deviation of the candidate's posterior output-quality estimation (i.e., Eq. 5) , λ is the exploration coefficient, ρ_i denotes the estimated execution success probability (i.e., Eq. 6), and D_i is the stable latency discount factor (i.e., Eq. 9). Candidates are ranked in descending order of U_i^{+}. This utility favors candidates with high estimated capability of the selected LLM and high execution reliability, while applying a controlled latency discount. The uncertainty bonus $\lambda\sigma_i$ encourages exploration of under-observed but potentially strong candidates.

2. Conservative Reliability Accumulation. Following the ranking, HACO incrementally constructs the hedge set using a pessimistic execution reliability estimate:

$$R_i^{-} = \min\left(0.98, \max(\epsilon_{\text{R}}, \mu_i - \gamma\sigma_i) \cdot \rho_i\right). \tag{11}$$

Here, $\gamma \geq 0$ controls the conservative margin in the capability score $\mu_i - \gamma\sigma_i$, and ρ_i weights this score by physical execution reliability. The lower bound ϵ_{R} stabilizes the capability term, while the upper cap 0.98 avoids perfect single-candidate reliability. This conservative estimate is used only for stopping, so that HACO does not terminate hedge construction based on overly optimistic reliability estimates.

Let P_{fail} denote the estimated cumulative failure probability that all currently selected candidates fail. Starting from $P_{\text{fail}} = 1$, HACO updates:

$$P_{\text{fail}} \leftarrow P_{\text{fail}}(1 - R_i^{-}) \tag{12}$$

whenever a new candidate is added. Redundancy allocation stops when

$$1 - P_{\text{fail}} \geq \tau, \tag{13}$$

or when the candidate pool is exhausted. A single shared estimate is insufficient for both purposes: an overly conservative utility estimate would suppress exploration, whereas an overly optimistic one would weaken the stopping guarantee.

In this way, Phase 2 decides both *which* candidates should enter the hedge set and *how many* to select under the conservative stopping rule. Appendix D.2 analyzes this rule under calibrated lower bounds and conditional independence, with an extension to correlated failure domains in Appendix D.3.

3.3 Phase 3: Hedged Execution

Given the hedge set selected, HACO launches all candidates in $\mathcal{S}$ in parallel and returns once the highest-utility successful candidate is determined. Remaining traces are harvested asynchronously for future profile updates. For each executed candidate $a_i \in \mathcal{S}$, let $s_i \in \{0, 1\}$ denote whether the execution succeeds, and let t_i denote its observed completion time. HACO denotes the execution-successful candidates observed for winner selection as

$$\mathcal{S}^+ = \{a_i \in \mathcal{S} \mid s_i = 1\}. \tag{14}$$

If $\mathcal{S}^+ = \emptyset$, HACO emits a role-failure payload to the controller for recovery and skips winner selection. Otherwise, HACO returns the candidate with the highest optimistic prioritization utility:

$$a_{\text{winner}} = \arg \max_{a_i \in \mathcal{S}^+} U_i^+. \tag{15}$$

Let $i^\star$ denote the index of a_{winner}. The observed invocation latency is the time when the selected winner can be determined:

$$T_{\text{obs}} = \max\left(t_{i^\star}, \max_{a_j \in \mathcal{S}: U_j^+ > U_{i^\star}^+} t_j^{\text{res}} \right), \tag{16}$$

where t_j^{res} denotes the resolution time of a higher-utility candidate, and the inner maximum is omitted when the set is empty. Thus, HACO ties blocking latency to the winner-decision time and uses the same U_i^+ signal for final selection, trading post-execution reranking for lower evaluation overhead.

3.4 Phase 4: Experience Harvesting

A key advantage of HACO is that redundancy is used not only for robust execution, but also for experience harvesting since all executed candidates in the hedge set could provide useful feedback for future allocation decisions.

Specifically, for each executed candidate $a_i \in \mathcal{S}_t$, HACO records its LLM-judge quality signal q_i, success status s_i, completion time t_i, token usage, internal failure status, and communication telemetry. Here, $\mathcal{S}_t$ denotes the hedge set at step t (and we write $\mathcal{S}$ when the step index is omitted). These traces are aggregated into two complementary profiles:

Candidate profiles. HACO updates uncertainty-aware capability estimate from LLM-judge quality signals. Here, q_i is used to update the Beta profile in Eq. (4) and changes the posterior mean and uncertainty (μ_i, σ_i) in Eq. (5). These updated estimates are reused in the next allocation step by the optimistic ranking score U_i^+ in Eq. (10) and the conservative reliability estimate R_i^- in Eq. (11). HACO also refreshes running statistics, e.g., call frequency, failure rate, latency, and token usage.

Network-link profiles. HACO logs inter-agent transmissions and updates link-level statistics, including average bandwidth, bandwidth jitter, and packet-loss rate. These statistics are reused to estimate physical reliability ρ_i in Eq. (6) and effective latency T_i in Eq. (7). The resulting latency cost and delay discount in Eqs. (8)–(9) then affect the next-round utility score and hedge-set construction.

This harvesting mechanism turns redundant execution into both immediate robustness and future allocation evidence. Each hedge set provides observations over agent capability and physical execution conditions, allowing HACO to refine agent selection and network-aware cost estimation over time. Algorithm 1 in Appendix D.1 summarizes the full procedure of HACO.

4 Experiments

4.1 Experimental Setting

Role types. HACO can be applied to any MAS that exposes role invocations and allows execution over multiple candidate agent instances. Here, we instantiate HACO on a multi-agent data-analysis system derived from DATAWISE. The underlying MAS is organized around four role types: planner, coder, debugger, and filter. These roles form a structured workflow in which the planner decomposes and updates task steps, the coder generates and executes code, the debugger repairs execution failures, and the filter cleans debugging outputs before execution resumes. This setting

provides a representative long-horizon agent controller with state transitions and self-debugging behavior, while leaving HACO independent of the specific controller implementation. The full MAS-DATAWISE workflow is described in Appendix C.1 and Fig. 8.

Execution environment. To evaluate runtime uncertainty, we build a heterogeneous execution environment that captures both network fluctuation and probabilistic candidate failures. Candidate agent instances are deployed across three execution zones, i.e., Global, Regional, and Local. Each zone contains LLM candidates instantiated through different model APIs. We configure candidate-level invocation failures and inter-zone network conditions, including latency, jitter, bandwidth, and loss rate, through a controllable interface for sensitivity and stress testing under heterogeneous deployment conditions. The motivation for using execution zones follows the cross-region inference patterns discussed in Appendix A (Table 2); concrete candidate-pool configurations are reported in Appendix C.2.2 (Tables 4–5), and the network simulator is detailed in Appendix C.2.1 (Fig. 9 and Table 6). We additionally evaluate HACO using Azure regional endpoints with measured latency and controlled failure injection (Appendix C.3, Table 7), and a separate measured-trace replay study (Appendix C.4).

Benchmarks. We evaluate HACO on three representative benchmarks. DSBench [28] evaluates data science agents on end-to-end workflows involving data preprocessing, feature engineering, and predictive modeling. InfiAgent-Bench [25] focuses on multi-step data analysis over structured CSV files in a code execution environment. MatplotBench [53] evaluates scientific visualization agents that generate plots from natural language specifications and structured data inputs. These benchmarks cover data-analysis reasoning, executable workflows, and visualization-oriented code generation.

Baselines. We compare HACO with representative baselines. Following recent LLM-routing benchmark practice [26], **Random** uniformly samples one candidate for execution, serving as an environment-agnostic single-route reference. Inspired by recent feedback-based LLM routing methods [36, 47], **BestOne** selects the best one candidate according to historical performance records. **UCB-Single** retains HACO's profiles, environment signals, UCB ranking and update rules but executes only the top-ranked candidate, providing a stronger single-route comparison. **MoA-style Best-of-N** [48] executes all candidate agents in parallel and uses an evaluator language model to score the candidate outputs, returning the highest-scoring response. Detailed descriptions and pseudocode are provided in Appendix D.4; hyperparameters and compute resources are reported in Appendices D.1 and C.5.

4.2 Experimental Results

Reliability under Runtime Uncertainty. Fig. 3 evaluates HACO under runtime degradation injected after task index 50 and under an Azure-backed bad-agent deployment. Across network degradation, candidate degradation, and real bad-agent deployment, BestOne shows the largest reliability collapse, with all-failed-rate increases of 72.8, 67.4, and 90.4 percentage points. The reason is that BestOne repeatedly favors the historically strongest candidate or zone, which becomes brittle once that execution path is degraded. Random is less concentrated, but it still lacks an explicit mechanism to identify and avoid degraded runtime conditions. HACO remains stable because redundant executions provide online evidence for reallocating future invocations. The selected-zone shifts in Fig. 3(d) show that HACO moves away from degraded regions, reducing Global selections by 80 and 40 percentage points under network and candidate degradation, respectively. MoA executes the full pool across all zones, so it has no corresponding pre-execution routing shift; its post-hoc winner-zone distributions are reported in Appendix E.4 and Table 11. In the real deployment, HACO shifts away from the failing US-labeled deployment and increases the use of JP and SE, while BestOne shifts further toward the failing region. Evaluator degradation has a minor effect, indicating that HACO does not strongly depend on the judge and can operate effectively with a weak LLM-as-judge evaluator. Overall, HACO approaches MoA-style robustness without full-pool execution, supporting the value of experience-guided adaptive redundancy under runtime uncertainty. A complementary Azure live/replay study uses a 1,200-call trace and evaluates 30 tasks with 24 logical candidates across four physical failure domains. HACO scores 65.00 versus 55.67 for UCB-Single at higher token and latency cost (Table 8); the protocol and phase results are reported in Appendix C.4.

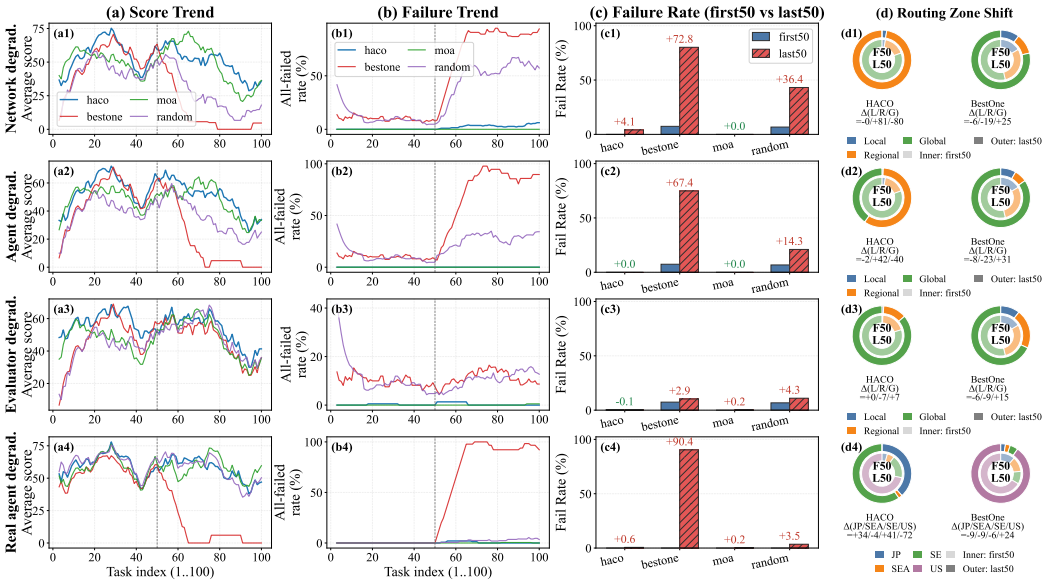


Figure 3: **Reliability evaluation under runtime degradation and real deployment failures.** Rows correspond to network degradation, candidate degradation, evaluator degradation, and a real-world Azure-backed deployment with injected candidate failures.

Table 1: UCB-Single comparison. Effectiveness is benchmark-specific; costs retain the recorded evaluation aggregates (Appendix E.1). Full metrics, including strict success, are in Table 12.

Benchmark	Method	Score/Acc.	Tokens (k)	Latency (s)	All-failed step rate (%)
MatplotBench	UCB-Single	51.66	42.18	46.11	1.52
	HACO	58.86	65.32	23.50	0.17
InfiAgent-Bench	UCB-Single	80.35	55.14	63.70	0.29
	HACO	89.01	91.04	31.31	0.37
DSBench	UCB-Single	33.28	112.63	270.35	0.00
	HACO	42.40	393.72	125.91	0.40

Overall Benchmark Performance. Fig. 4 compares HACO with single-route, fixed-zone, and full-redundancy baselines under the heterogeneous execution environment. All axes are normalized so that larger values indicate better performance, including effectiveness, strict success, token efficiency, latency efficiency, and all-failed-step reliability. Formal definitions of these radar metrics are provided in Appendix E.1. Additional raw-result analyses are provided in Fig. 10. Across all benchmarks, HACO forms one of the largest and most balanced radar profiles, indicating consistent gains in task effectiveness, execution reliability, and practical efficiency. Single-route baselines expose the limitation of committing to one candidate before execution: Random is inexpensive but unstable, while BestOne can exploit historical performance but remains sensitive to unreliable candidates or execution zones. Fixed-zone baselines further show that no single execution zone dominates across all benchmarks and metrics. MoA-style Best-of-N provides strong reliability through full-pool execution, but this comes with substantially higher token and latency cost. In contrast, HACO uses an adaptive hedge subset, achieving reliability close to MoA while avoiding exhaustive execution. Table 1 compares HACO with the stronger UCB-Single baseline. HACO improves effectiveness on all three benchmarks, using more tokens and less latency; all-failed rates vary by benchmark.

Table 12 in Appendix E reports the full results. Appendices E.5 and E.6 discuss fixed-global results and external-reference estimates (Table 13). Overall, these results support the main claim that adaptive redundancy provides a better effectiveness–efficiency–reliability trade-off than either single routing or full parallel execution.

Behavioral Analysis. Fig. 5 examines HACO's adaptive redundancy and routing behavior. Unlike Random, BestOne, and fixed-zone baselines, which execute one candidate per invocation, and unlike

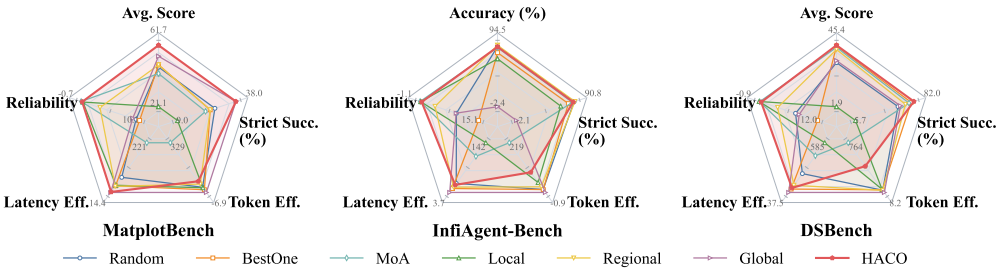


Figure 4: **Performance comparison under the heterogeneous execution environment.** Results are reported on MatplotBench, InfiAgent-Bench, and DSBench for Random, BestOne, MoA-style Best-of-N, fixed-zone Local, fixed-zone Regional, fixed-zone Global, and HACO.

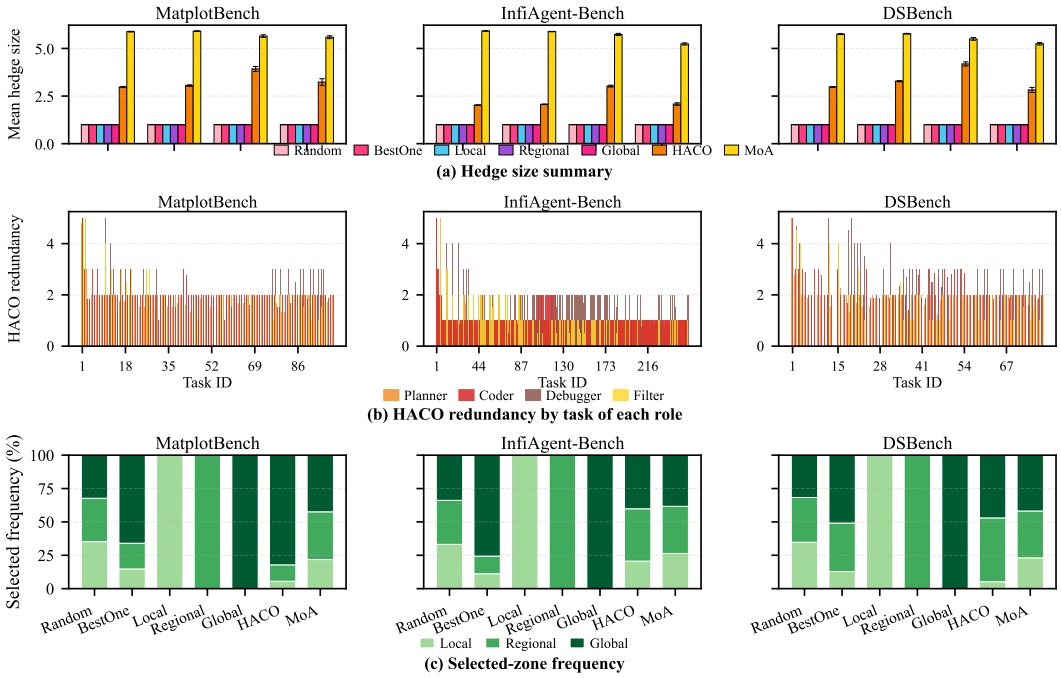


Figure 5: **HACO behavior.** (a) Mean hedge size of baselines and HACO, (b) per-task role redundancy, and (c) selected-zone frequency across three benchmarks.

MoA-style baseline, which executes the full candidate pool, HACO selects a hedge set whose size varies adaptively to adjust redundancy according to task difficulty and runtime uncertainty, as shown in Fig. 5 (a). The per-task traces in Fig. 5 (b) show role- and dataset-dependent allocation patterns, with harder workflows such as DSBench generally requiring larger hedge sets than simpler analysis tasks such as InfiAgent-Bench. The selected-zone distribution in Fig. 5 (c) shows that HACO uses multiple execution zones instead of committing to a single fixed zone. These results support HACO's design goal of treating redundancy and execution environment as runtime decision variables.

Reliability–Cost Trade-off. Fig. 6 (a)–(c) show that the reliability target τ provides a controllable knob for balancing task quality, execution reliability, and token cost. As τ increases from 0.70 to 0.95, the step failure rate decreases from 1.87% to 0.17%, while the average score increases from 53.77 to 58.86. This improvement comes at the cost of more redundant execution, as reflected by the increasing token mean in Fig. 6 (c). The score-token curve changes smoothly, indicating that HACO can move gradually between low-cost and high-reliability operating points. Single-route methods operate in a low-cost but less reliable regime, while MoA improves reliability through full-pool execution. HACO occupies an intermediate region, approaching high reliability without always paying the full cost of exhaustive parallelism. These results support the claim that adaptive redundancy improves robustness and exposes a tunable performance-efficiency frontier. Appendix E.2 and Table 9 report aggregate conservativeness and target reachability at these operating points.

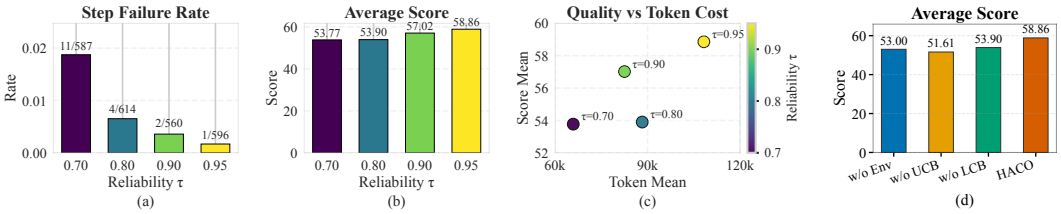


Figure 6: **Reliability-target sweep and component ablation in MatplotBench.** Panels (a)–(c) sweep the system reliability target $\tau \in \{0.70, 0.80, 0.90, 0.95\}$ and report step failure rate, average score, and the score-token relationship. Panel (d) compares HACO with ablated variants that remove environment-aware modeling (Env), the uncertainty bonus (UCB), or conservative stopping (LCB).

Component Ablation Fig. 6 (d) evaluates the contribution of three key components in HACO on MatplotBench. Removing any component decreases the average score, indicating that HACO's gain does not come from a single heuristic. Without environment-aware modeling, the score drops from 58.86 to 53.00, showing that effective candidate selection requires communication reliability, latency, and output quality to be modeled jointly. Among the three variants in Fig. 6(d), removing the uncertainty bonus gives the lowest score, 51.61, suggesting that optimistic exploration is important for discovering under-observed but useful candidates. Replacing conservative stopping also reduces the score to 53.90, which suggests that pessimistic reliability estimation helps avoid premature termination of the hedge set. Overall, the ablation confirms that HACO benefits from the joint design of environment-aware routing, uncertainty-aware ranking, and conservative reliability accumulation. Controlled variants test candidate-specific attribution, fixed-K allocation ($K = 3$) and top-1 execution on both MatplotBench and InfiAgent-Bench. All six variants reduce mean effectiveness on both datasets (Table 10); their definitions are given in Appendix E.3.

5 Conclusion

We presented HACO, a hedged agent computing approach for LLM-based agent systems. HACO views each role invocation as a runtime decision that jointly depends on role type, LLM choice, and execution environment, instead of treating it as a fixed call determined only by a role–agent instance pair. It combines query-adaptive redundant execution with experience harvesting, so that redundant invocations improve current robustness while also providing feedback for future allocation. Across agent benchmarks and stress settings, HACO improves execution reliability and output quality under heterogeneous runtime conditions, including network degradation, candidate degradation, evaluator degradation, and an Azure-backed deployment with injected candidate failures. HACO reduces the fragility of single-route execution and approaches the robustness of full parallel execution without always incurring its full token and latency cost. These results suggest that execution-aware redundancy allocation is a practical mechanism for building reliable LLM-based agent systems in dynamic real-world deployment environments. Limitations and broader impacts are discussed in Appendices F and G.

Acknowledgments and Disclosure of Funding

The authors received no external funding for this work and declare no competing interests.

References

[1] Amazon Web Services. Increase throughput with cross-region inference. `https://docs.aws.amazon.com/bedrock/latest/userguide/cross-region-inference.html`, 2026. Accessed: 2026-03-13.

[2] Amazon Web Services. Geographic cross-region inference. `https://docs.aws.amazon.com/bedrock/latest/userguide/geographic-cross-region-inference.html`, 2026. Accessed: 2026-03-13.

[3] Amazon Web Services. Regional availability - amazon bedrock. https://docs.aws.amazon.com/bedrock/latest/userguide/models-region-compatibility.html, 2026. Accessed: 2026-03-13.

[4] Anthropic. Effective harnesses for long-running agents. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. Published Nov. 26, 2025.

[5] Yuda Bi, Ying Zhu, and Vince D. Calhoun. Redundancy as a structural information principle for learning and generalization. *arXiv preprint arXiv:2510.10938*, 2025. URL https://arxiv.org/abs/2510.10938.

[6] Mark Carson and Darrin Santay. Nist net: A linux-based network emulation tool. *Computer Communication Review*, 33(3):111–126, 2003.

[7] Mingju Chen, Guibin Zhang, Heng Chang, Yuchen Guo, and Shiji Zhou. A-mapreduce: Executing wide search via agentic mapreduce. *arXiv preprint arXiv:2602.01331*, 2026.

[8] China Academy of Information and Communications Technology and Alibaba Cloud Computing Co., Ltd. Guiding principles for excellent generative ai architecture design, September 2025. URL https://pdf.dfcfw.com/pdf/H3_AP202509181745854315_1.pdf. Accessed: 2026-04-07.

[9] Cloudflare. Cloudflare global network | data center locations. https://www.cloudflare.com/network/, 2026. Accessed: 2026-03-13.

[10] Cloudflare. Cloudflare workers ai overview. https://developers.cloudflare.com/workers-ai/, 2026. Accessed: 2026-03-13.

[11] CNBC. Microsoft says azure cloud computing service disrupted by fiber cuts in the red sea. https://www.cnbc.com/2025/09/06/microsoft-azure-cloud-computing-service-disrupted-red-sea-fiber-cuts.html, 2025. Accessed: 2026-05-02.

[12] CNN Business. Red sea cables have been damaged, disrupting internet traffic. https://www.cnn.com/2024/03/04/business/red-sea-cables-cut-internet, 2024. Accessed: 2026-05-02.

[13] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

[14] Zhaopeng Feng, Liangcai Su, Zhen Zhang, Xinyu Wang, Xiaotian Zhang, Xiaobin Wang, Runnan Fang, Qi Zhang, Baixuan Li, Shihao Cai, et al. Agentswing: Adaptive parallel context management routing for long-horizon web agents. *arXiv preprint arXiv:2603.27490*, 2026.

[15] Silvia Ferrari and Francisco Cribari-Neto. Beta regression for modelling rates and proportions. *Journal of applied statistics*, 31(7):799–815, 2004.

[16] Fireworks AI. Regions - fireworks ai docs. https://docs.fireworks.ai/deployments/regions, 2026. Accessed: 2026-03-13.

[17] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. URL https://arxiv.org/abs/2411.04468.

[18] Google Cloud. Deployments and endpoints. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/learn/locations, 2026. Accessed: 2026-03-13.

[19] Google Cloud. Standard paygo. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/standard-paygo, 2026. Accessed: 2026-03-13.

[20] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. Kollaps: Decentralized and dynamic topology emulation. *arXiv preprint arXiv:2004.02253*, 2020.

[21] Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. Language model cascades: Token-level uncertainty and beyond. *arXiv preprint arXiv:2404.10136*, 2024.

[22] Junda He, Christoph Treude, and David Lo. LLM-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.

[23] Stephen Hemminger. *tc-netem(8) Linux Manual Page*, 2005. Linux Network Emulator (NetEm).

[24] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. URL `https://arxiv.org/abs/2308.00352`.

[25] Xueyu Hu, Ziyu Zhao, Shuang Wei, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. In *International Conference on Machine Learning*, 2024.

[26] Zhongzhan Huang, Guoming Ling, Yupei Lin, Yandong Chen, Shanshan Zhong, Hefeng Wu, and Liang Lin. RouterEval: A comprehensive benchmark for routing LLMs to explore model-level scaling up in LLMs. *arXiv preprint arXiv:2503.10657*, 2025.

[27] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14165–14178, 2023.

[28] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents from becoming data science experts? In *International Conference on Learning Representations*, 2025.

[29] Meelis Kull, Telmo Silva Filho, and Peter Flach. Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers. In *Artificial intelligence and statistics*, pages 623–631. PMLR, 2017.

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

[31] LiteLLM. Auto routing - litellm. `https://docs.litellm.ai/docs/proxy/auto_routing`, 2026. Accessed: 2026-03-13.

[32] Yizhuo Ma, Ke Qin, and Shuang Liang. Beta-lr: Interpretable logical reasoning based on beta distribution. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1945–1955, 2024.

[33] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

[34] Microsoft. Understanding deployment types in microsoft foundry models. `https://learn.microsoft.com/en-us/azure/foundry/foundry-models/concepts/deployment-types`, 2026. Accessed: 2026-03-13.

[35] Microsoft Azure. Post incident review: Thermal event impacting multiple services – west europe. `https://azure.status.microsoft/en-us/status/history/?force_isolation=true`, 2025. Tracking ID: 2LGD-9VG. Accessed: 2026-05-02.

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

[37] OpenAI. Data controls in the openai platform. `https://developers.openai.com/api/docs/guides/your-data/`, 2026. Accessed: 2026-03-13.

[38] OpenRouter. Intelligent multi-provider request routing. `https://openrouter.ai/docs/guides/routing/provider-selection`, 2026. Accessed: 2026-03-13.

[39] OpenRouter. Openrouter quickstart guide. `https://openrouter.ai/docs/quickstart`, 2026. Accessed: 2026-03-13.

[40] Portkey. Fallbacks - portkey docs. `https://portkey.ai/docs/product/ai-gateway/fallbacks`, 2026. Accessed: 2026-03-13.

[41] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. *ACL*, 2024. URL `https://arxiv.org/abs/2307.07924`.

[42] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

[43] Shaina Raza, Ranjan Sapkota, Manoj Karkee, and Christos Emmanouilidis. Trism for agentic Trism: A review of trust, risk, and security management in LLM-based agentic multi-agent systems. *AI Open*, 2026.

[44] Peter Reichl, Sebastian Egger, Raimund Schatz, and Alessandro D'Alconzo. The logarithmic nature of qoe and the role of the weber-fechner law in qoe assessment. In *2010 IEEE international conference on communications*, pages 1–5. IEEE, 2010.

[45] Philipp Schmid. The importance of agent harness in 2026. Blog post, 2026. URL `https://www.philschmid.de/agent-harness-2026`.

[46] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. SLoRA: Scalable serving of thousands of LoRA adapters. *Proceedings of Machine Learning and Systems*, 6:296–311, 2024.

[47] Asterios Tsiourvas, Wei Sun, and Georgia Perakis. Causal LLMs routing: End-to-end regret minimization from observational data. *arXiv preprint arXiv:2505.16037*, 2025.

[48] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y. Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, 2025.

[49] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[50] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

[51] Bingyu Yan, Zhibo Zhou, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, Zhoujun Li, Chaozhuo Li, and Xiaoming Zhang. Beyond self-talk: A communication-centric survey of LLM-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025.

[52] Yingxuan Yang, Chengrui Qu, Muning Wen, Laixi Shi, Ying Wen, Weinan Zhang, Adam Wierman, and Shangding Gu. Understanding agent scaling in llm-based multi-agent systems via diversity. *arXiv preprint arXiv:2602.03794*, 2026. URL `https://arxiv.org/abs/2602.03794`.

[53] Zhiyu Yang, Zihan Zhou, Shuo Wang, et al. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. In *Findings of ACL*, 2024.

[54] Ziming You, Yumiao Zhang, Dexuan Xu, Yiwei Lou, Yandong Yan, Wei Wang, Huaming Zhang, and Yu Huang. Datawiseagent: A notebook-centric llm agent framework for automated data science. *arXiv preprint arXiv:2503.07044*, 2025.

[55] Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *arXiv preprint arXiv:2310.03094*, 2023.

[56] Xuechen Zhang, Zijian Huang, Ege Onur Taga, Carlee Joe-Wong, Samet Oymak, and Jiasi Chen. Efficient contextual LLM cascades through budget-constrained policy learning. *Advances in Neural Information Processing Systems*, 37:91691–91722, 2024.

[57] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

[58] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.

A Cross-Region Inference in Practice

This section documents publicly available evidence that cross-region inference is already a practical deployment pattern in modern LLM serving systems. Modern platforms expose several distinct abstractions that decouple API ingress from the physical location of inference execution. Although their implementations differ, these systems consistently allow the execution region to vary with deployment configuration, routing policy, capacity conditions, or compliance constraints.

From a systems perspective, existing platforms can be grouped into three representative categories.

(1) Cloud-region routing platforms Some LLM serving systems expose global endpoints that dynamically route requests across multiple cloud regions. In these systems, the developer interacts with a stable API endpoint while the provider determines the actual inference location according to internal routing policies such as capacity availability or load balancing.

Azure Foundry provides three deployment types: *Global*, *Data Zone*, and *Regional*. Under Global deployment, requests may be processed in any Azure region where the model is available. Data Zone deployments restrict processing within a specified geography such as the US or EU, while Regional deployments keep execution within a specific region [34].

Vertex AI exposes a similar abstraction through *regional endpoints* and a *global endpoint*. The global endpoint draws from a larger multi-region capacity pool and therefore offers higher availability and burst tolerance than a single region. However, the platform does not expose the actual processing region of a request to the developer [18, 19].

Amazon Bedrock provides cross-region execution through *inference profiles*. These profiles support both *geographic cross-region inference*, which restricts execution within a specific geography, and *global cross-region inference*, which allows requests to be routed across commercial AWS regions worldwide [1–3]. Bedrock also documents that the final inference region can be observed through service telemetry such as CloudTrail.

In all of these systems, region selection is determined by the infrastructure layer.

(2) Region pinning and data residency A second design treats region as an explicit configuration parameter. In this setting, developers choose the processing region during deployment, and inference is performed within that region.

OpenAI provides regional processing and data residency features at the project level. Developers select a project region and, where supported, access the corresponding regional endpoint [37]. While

Table 2: Publicly documented cross-region inference mechanisms in representative LLM serving platforms. Existing systems already treat execution location as an operational variable at the infrastructure layer.

Platform	Public cross-region support	Developer control / visibility	Primary platform objective
Azure Foundry	Global, Data Zone, and Regional deployments. Global and Data Zone deployments may route requests across eligible datacenters.	Developers choose the deployment type. Global deployments may process prompts and responses in any eligible Azure region, while Regional deployments keep processing in the deployment region.	Increase quota, throughput, model availability, and geography-level compliance.
Vertex AI	Regional endpoints and a global endpoint. The global endpoint routes traffic over a larger multi-region capacity pool.	Developers choose regional versus global endpoints, but the actual processing region of the global endpoint is not exposed to the user.	Improve availability, reduce 429 errors, and absorb bursty demand through capacity-aware routing.
Amazon Bedrock	Geographic and Global cross-region inference profiles route requests across multiple AWS Regions.	Developers choose an inference profile. The actual inference region can be observed through CloudTrail.	Increase throughput, absorb traffic bursts, and satisfy geography-level compliance constraints.
OpenAI API	Regional processing and data residency at the project level.	Developers explicitly choose a project region and, where supported, use the corresponding regional endpoint.	Support compliance, regional processing, and in some cases lower latency.
Cloudflare Workers AI	Inference runs on Cloudflare's global network with distributed GPU deployment.	Developers use a global network abstraction, while the concrete inference region is left unspecified.	Serve requests closer to users and improve responsiveness and resilience.
Fireworks AI	Default multi-region deployments over `GLOBAL`, `US`, `EUROPE`, and `APAC`. Deployments may run in any region within the selected scope.	Developers can use a multi-region deployment or pin the deployment to a single region.	Elastic scaling, availability, regional compliance, and deployment near users.

this model does not expose dynamic cross-region routing in the same way as Azure, Vertex AI, or Bedrock, it confirms that physical processing location is treated as an operational concern in production LLM APIs.

(3) Distributed inference infrastructure A third design emerges in systems that provide inference over geographically distributed GPU infrastructure.

Cloudflare Workers AI runs inference on GPUs deployed across Cloudflare's global edge network, allowing models to execute close to user locations [10, 9]. Fireworks provides multi-region deployments with scopes such as `GLOBAL`, `US`, `EUROPE`, and `APAC`. Deployments may run in any region within the selected multi-region scope [16]. In both cases, physical location is treated as part of the serving substrate, not as a fixed endpoint.

Across these designs, the physical execution region of an LLM request is therefore not fixed *a priori*. Instead, it may vary with deployment configuration, routing policy, capacity conditions, or compliance constraints. This observation directly supports our formulation in which the execution environment of a role invocation represents a runtime context with potentially different latency, execution reliability, and downstream connectivity characteristics.

Table 2 shows that cross-region inference is already supported by major model serving platforms. As we can see, current cloud inference platforms already consider region, but they do so mainly for infrastructure-level goals such as throughput, burst absorption, availability, and compliance [34, 18, 19, 1, 2, 37, 10, 16]. By contrast, third-party gateways and model access layers usually expose LLM choice or provider choice as the main control surface, emphasizing fallback, load balancing, and provider-level routing, with less direct attention to the execution region itself [39, 38, 40, 31]. What remains weakly addressed is *request-aware, region-aware execution* for agent workloads. Task-level performance depends not only on LLM choice, but also on where inference runs. HACO

introduces an application-level coordination layer for agent execution. It operates above existing provider routing mechanisms and makes task-level decisions about whether to launch redundant executions, which candidates to include, and how to balance latency, system reliability, and output quality. This design incorporates query characteristics and execution region conditions into the coordination process, enabling agent systems to utilize regional diversity during inference.

B Comparison with Related Runtime and Inference-Time Paradigms

Table 3: HACO compared with related runtime and inference-time paradigms.

Paradigm	**Control point**	**Execution pattern**	**Relation to HACO**
LLM routing [13, 36]	Single-executor selection	One model or endpoint per request	Routing chooses one executor for a request, usually to balance quality and cost. HACO operates at the role-invocation allocation point and selects an adaptive hedge set when a single execution is not sufficient for the target reliability level.
Fallback or cascade [55, 21, 56]	Post-hoc escalation or recovery	Sequential backup calls	Fallback and cascade methods invoke a stronger or backup executor after failure, low confidence, or insufficient consistency. HACO allocates redundancy before execution using estimated capability, failure, latency, and link conditions.
Batch inference [30, 58, 46]	Serving-batch construction	Many requests processed together	Batch inference improves serving throughput or hardware utilization across requests. HACO controls reliability for each role invocation before the launched calls are served.
Multi-output aggregation [27, 48, 49]	Output selection or synthesis	Multiple samples, agents, or models	Best-of-N, self-consistency, MoA, and ensemble methods mainly improve final answer quality through selection, voting, ranking, or synthesis. HACO decides which executions to launch under runtime uncertainty; aggregation can be applied after the hedge set produces outputs as a potential future research direction.
HACO	**Role-invocation allocation**	**Adaptive parallel execution**	**HACO selects a reliability-constrained hedge set over candidate agent instances and stops allocation when the target reliability level is reached.**

Table 3 clarifies the control point addressed by HACO. Existing paradigms typically operate at single-route selection, post-failure recovery, serving-batch construction, or output aggregation. HACO instead addresses the preceding allocation problem for each role invocation. Specifically, before execution, it decides which candidate agent instances should be activated and when the selected hedge set has reached the required reliability level. This formulation makes output aggregation a possible downstream policy.

Role of Redundancy in HACO. Figure 7 further clarifies the role of redundancy in HACO. In contrast to methods that use parallelism mainly for post-hoc output selection, synthesis, or serving efficiency, HACO uses redundancy as a runtime control mechanism at the role-invocation level. For the current invocation, hedged execution improves reliability by tolerating candidate failures, high latency, and unstable service or network conditions. For future invocations, the executed hedge set provides additional traces for experience harvesting, including quality, success or failure, latency, and network statistics. These observations update candidate and link profiles, allowing HACO to improve later routing and hedge-set allocation. Thus, redundancy in HACO serves both as an immediate protection mechanism and as a data-collection mechanism for adaptive future decisions.

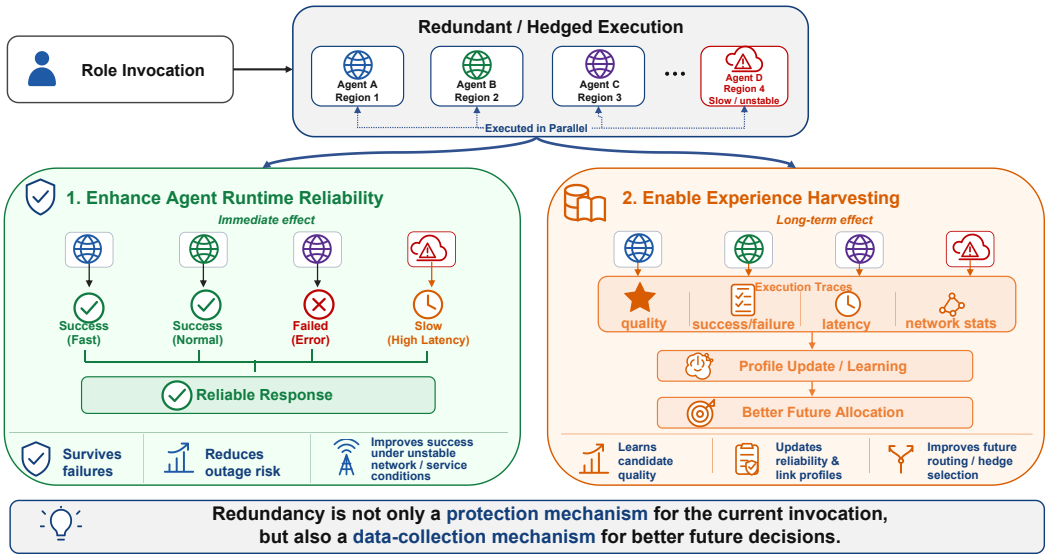


Figure 7: **Role of redundancy in HACO.** Redundant or hedged execution improves reliability for the current role invocation while also collecting execution traces for future allocation. HACO records quality, success/failure, latency, and network statistics from executed candidates, updates candidate and link profiles, and uses these profiles to improve later routing and hedge-set selection.

C Experimental Setup Details

C.1 MAS-DATAWISE settings

Our multi-agent system is implemented as an extension of DATAWISE. In particular, we build on the original DATAWISE [54] framework and adapt it from a single-system workflow into a collaborative multi-agent setting, while preserving its controller logic and execution structure. We use this system as the primary testbed in our experiments because it provides a representative controller with explicit state transitions, iterative code execution, and built-in self-debugging behaviors. At the same time, HACO is not tied to this specific controller design. Its routing and redundancy allocation mechanisms are agnostic to the internal orchestration policy of the underlying MAS, and can therefore be applied not only to complex finite-state-machine (FSM) controllers such as DATAWISE, but also to simpler workflow-style controllers such as METAGPT [24], as well as other controller paradigms.

Fig. 8 illustrates the DATAWISE concurrent FSM workflow from a role-oriented perspective. The underlying controller is a six-state finite-state machine with states INITIATE_PLAN, APPEND_CODE, DEBUG, FILTER, UPDATE_PLAN, and TERMINATE. Functionally, however, the workflow is organized around four role types. The planner initializes and updates task steps, the coder performs iterative code generation and execution, the debugger repairs failures arising during execution or verification, and the filter cleans debugging outputs before execution resumes. The coding stage is controlled by explicit action signals: AWAIT_TAG prolongs the current execution stage, whereas completion signals return control to the planner. At the workflow level, the planner determines whether to iterate on the current step, advance to the next step, or terminate the process through the signals ITERATE_ON_LAST_STEP, ADVANCE_TO_NEXT_STEP, and FULFILL_INSTRUCTION. Together, these mechanisms form a structured self-debugging workflow in which recovery follows the path DEBUG $\rightarrow$ FILTER $\rightarrow$ APPEND_CODE.

C.2 Simulating Environment

Fig. 9 provides an overview of the simulated experimental setup. It combines the zone-aware network topology over Local, Regional, and Global execution zones with the role-specific candidate pool used by the MAS. The network side specifies latency, jitter, bandwidth, and loss for each logical inter-zone link, while the candidate side shows the LLM backbone, replica count, failure rates,

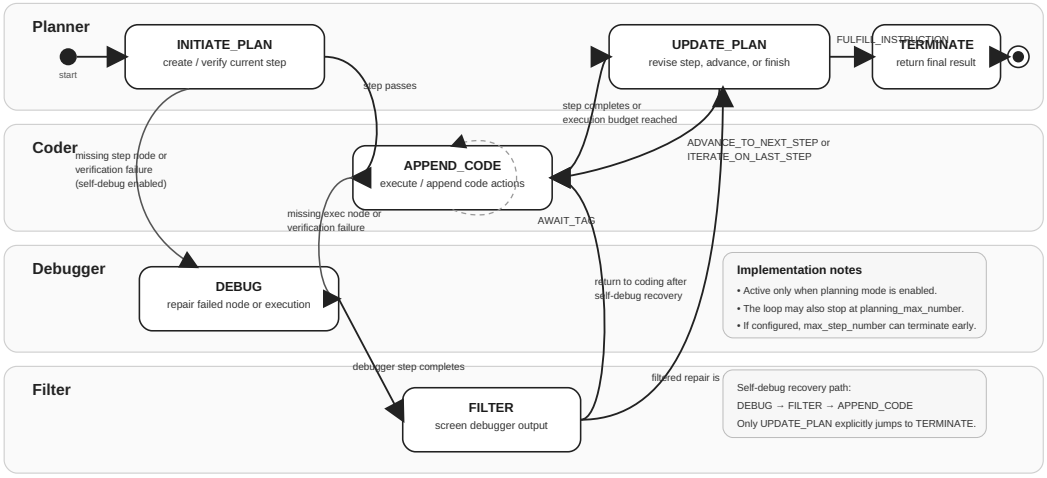


Figure 8: **Role-oriented view of the DATAWISE concurrent FSM workflow.** The underlying controller is a six-state finite-state machine, while the execution dynamics can be organized into four role types: planner, coder, debugger, and filter. The figure highlights role collaboration and the self-debug recovery path.

and temperature settings for each role. The following two sections detail these two components: the topology-based network simulator and the candidate-pool configuration.

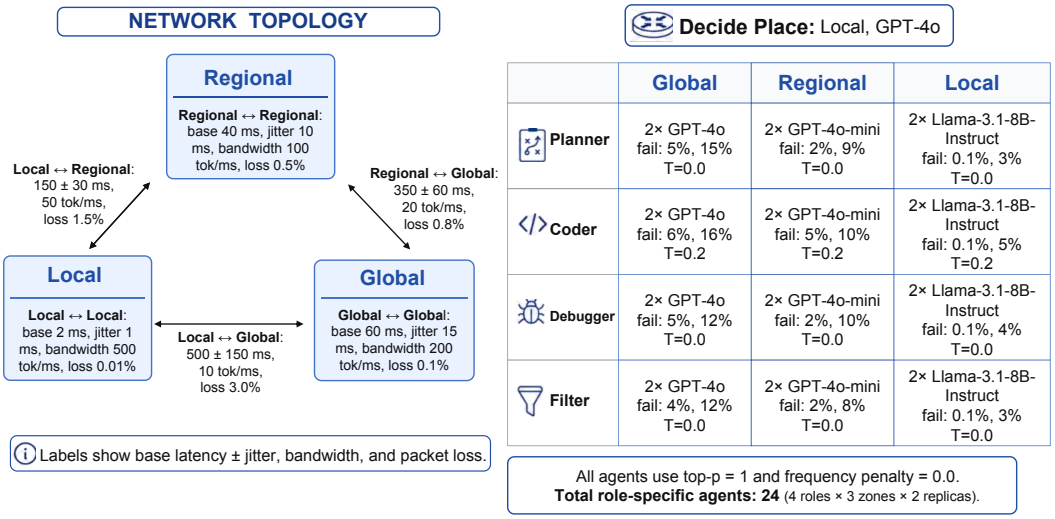


Figure 9: **Simulating environment overview.**

C.2.1 Network Simulation Details

We simulate inter-candidate communication using a **message-level network model**. Specifically, we directly approximate the end-to-end latency and failure behavior of transmitting a complete message (e.g., a list of notebook cells) between candidates. This design follows the parameterized network-emulation paradigm used in prior systems such as NIST Net and Linux NetEm, which models the network behavior using delay, jitter, bandwidth constraints, and loss parameters [6, 23]. It is also consistent with the view that, for distributed applications, the most important network properties are emergent end-to-end characteristics such as latency, bandwidth, packet loss, and jitter, while the full internal state of routers and switches is less directly relevant [20].

The topology in Fig. 9 and Table 6 is reported in simulator configuration units: latency in milliseconds and bandwidth in tokens/ms. The simulator computes message-level communication delay using these units. Before the resulting observations are used by HACO's effective-latency model

in Eq. (7), payload size is represented in bytes, bandwidth statistics are converted to bytes/s, and latency values are converted to seconds.

Network topology. We define a set of execution zones

$$\mathcal{Z} = \{\texttt{Global}, \texttt{Regional}, \texttt{Local}\}.$$

Each execution-zone pair is associated with a logical link whose properties are specified in a topology matrix. Each link is parameterized by: (i) base latency μ in milliseconds, (ii) jitter standard deviation σ in milliseconds, (iii) bandwidth B in tokens/ms, and (iv) loss rate $p \in [0, 1]$.

Message size. Given a message at step i consisting of a list of notebook cells, we estimate its payload size $M^{(i)}$ (in bytes) by serializing the cells into JSON.

Transmission delay. For a successful transmission at step i, the serialization/transmission delay is

$$T_{\text{tx}}^{(i)} = \frac{M^{(i)}}{4B}, \tag{17}$$

where B is the configured bandwidth in tokens/ms and we assume 1 token ≈ 4 bytes.

Path delay with jitter. The path delay is modeled as a base latency perturbed by zero-mean Gaussian noise:

$$T_{\text{path}}^{(i)} = \max\Big(T_{\min}(\mu),\ \mu + \xi^{(i)}\Big), \qquad \xi^{(i)} \sim \mathcal{N}(0, \sigma^2), \tag{18}$$

where $T_{\min}(\mu)$ is a latency floor that prevents unrealistic near-zero delays under negative jitter realizations. In particular, we use a *base-relative* lower bound:

$$T_{\min}(\mu) = \max(\epsilon_{\text{floor}},\ \zeta\mu), \tag{19}$$

where ϵ_{floor} is a small absolute lower bound and $\zeta \in (0, 1)$ is a fixed ratio. In our implementation, we use $\epsilon_{\text{floor}} = 1$ ms and $\zeta = 0.2$. This design preserves the scale differences between short-range and long-range links: for example, local links may have lower absolute floors than cross-region links, while all links remain protected against invalid near-zero latency samples.

Communication failure and timeout. We model failures at the *message level*. For each transmission at step i, a failure occurs independently with probability p. We abstract away lower-layer retransmission and transport protocol dynamics, and approximate the *end effect* of unsuccessful communication at the application level: a failed transmission incurs a fixed timeout penalty,

$$t_{\text{net}}^{(i)} = T_{\text{timeout}}, \tag{20}$$

where $T_{\text{timeout}} = 10$ seconds in our implementation. This corresponds to application-level request timeout behavior, such as an RPC timeout, and does not model packet-level network timeout.

Total communication latency. Otherwise (i.e., if the transmission succeeds), the communication latency is computed as

$$t_{\text{net}}^{(i)} = T_{\text{tx}}^{(i)} + T_{\text{path}}^{(i)}. \tag{21}$$

Equivalently, the complete model can be written as

$$t_{\text{net}}^{(i)} = \begin{cases} T_{\text{timeout}}, & \text{with probability } p, \\ T_{\text{tx}}^{(i)} + T_{\text{path}}^{(i)}, & \text{with probability } 1 - p, \end{cases} \qquad \xi^{(i)} \sim \mathcal{N}(0, \sigma^2). \tag{22}$$

Discussion. Our simulator is intentionally lightweight. It does not model packet-level mechanisms such as retransmissions, congestion control, reordering, or queueing disciplines. Instead, these effects are abstracted through stochastic delay and application-level timeout penalties. The use of a base-relative latency floor further ensures that jitter perturbs links realistically without collapsing long-range links into implausibly small delays. This abstraction is sufficient for our setting, which focuses on how heterogeneous inter-zone communication affects multi-agent coordination, while leaving exact transport-protocol behavior outside the scope of the simulator.

C.2.2 Candidate Pool Details

This section details the candidate-pool configurations used in the experiments. The implementation loads role-specific candidates from the experiment configuration and uses the active network-topology block for the topology-based network simulator. The descriptions below focus on the resulting runtime settings and omit configuration-file layout details.

Configurations for reliability under runtime uncertainty. The reliability-under-runtime-uncertainty experiments in Fig. 3 evaluate the degradation settings discussed in Sec. 4.2: network degradation, candidate degradation, and evaluator degradation. Each simulated setting changes one runtime dimension after task index 50 while preserving the same role-invocation interface and MAS workflow. Table 4 summarizes how each stress setting differs from the overall benchmark performance configuration.

Table 4: Configuration changes used for reliability evaluation under runtime uncertainty. Each stress setting changes one axis of the runtime system: candidate availability, network connectivity, or evaluator capability.

Stress setting	**Runtime setup**	**Change relative to the overall benchmark performance configuration**
Candidate degradation	Global candidate outage	Keeps the default network topology, but forces selected high-capability Global candidates to fail. In particular, both Global planner replicas and one Global coder replica are assigned failure rate 1.0. All other candidate definitions and failure rates remain the same as in the overall benchmark performance pool.
Network degradation	Global-link outage	Keeps the same 24 candidate definitions and candidate failure rates, but changes the active network topology. All links involving the Global zone are made unavailable with loss rate 1.0: Local–Global, Regional–Global, and Global–Global. Local–Regional and Regional–Regional remain reachable with 60 ms / 10 ms / 150 tokens per ms / 0.003 loss and 15 ms / 4 ms / 220 tokens per ms / 0.0008 loss, respectively.
Evaluator degradation	Weak router/evaluator	Keeps the overall benchmark performance candidate pool and default network topology, but replaces the router/evaluator model with a Local `meta-llama/llama-3.1-8b-instruct` instance. This setting stresses methods whose final selection depends on evaluator quality, especially full parallel generation followed by post-hoc scoring.

Configuration for overall benchmark performance. The overall benchmark performance results in Fig. 4 compare HACO with single-route, fixed-zone, and full-redundancy baselines under the heterogeneous execution environment described in the main text. This setting uses the full heterogeneous candidate pool. The pool contains four role types: planner, coder, debugger, and filter. Each role has six candidate agent instances: two Global candidates using `gpt-4o`, two Regional candidates using `gpt-4o-mini`, and two Local candidates using `meta-llama/llama-3.1-8b-instruct`. Thus, each execution zone contributes two candidates per role and eight candidates in total across the four roles, yielding 24 role candidates in the full pool. The router/evaluator agent is placed in the Local zone and uses `gpt-4o`; it supports routing and Best-of-N output scoring, but is not counted as a role candidate in the pool size. All candidates use `top_p`=1 and frequency penalty 0. The planner, debugger, and filter roles use temperature 0.0, while the coder role uses temperature 0.2. Table 5 summarizes the simulated candidate pool used for the overall benchmark performance setting.

For the main comparison, Random and BestOne select a single candidate from this pool, MoA executes the full pool for the invoked role, and HACO selects a hedge subset according to its reliability target. The fixed-zone baselines use restricted variants of the same role structure. The Local baseline keeps one Local Llama candidate per role with failure rate 0.001. The Regional baseline keeps one Regional `gpt-4o-mini` candidate per role with failure rate 0.02. The Global baseline keeps one Global `gpt-4o` candidate per role with failure rate 0.05. Therefore, each fixed-zone baseline uses four role candidates in total, all from a single execution zone. These fixed-zone configurations use the same default simulated network topology as Table 6.

Table 5: Simulated candidate pool for overall benchmark performance. Each failure rate corresponds to one replica in the same role-zone-model group.

Role	Global candidates `gpt-4o`	Regional candidates `gpt-4o-mini`	Local candidates `Llama-3.1-8B`	Temp.
Planner	Failure rates 0.05, 0.15	Failure rates 0.02, 0.09	Failure rates 0.001, 0.03	0.0
Coder	Failure rates 0.06, 0.16	Failure rates 0.05, 0.10	Failure rates 0.001, 0.05	0.2
Debugger	Failure rates 0.05, 0.12	Failure rates 0.02, 0.10	Failure rates 0.001, 0.04	0.0
Filter	Failure rates 0.04, 0.12	Failure rates 0.02, 0.08	Failure rates 0.001, 0.03	0.0

Table 6: Default simulated network topology. The simulator interprets each entry as a logical inter-zone link with stochastic delay, bandwidth, and message-level loss.

Link	Base latency (ms)	Latency Jitter (ms)	Bandwidth (tokens/ms)	Loss rate
Local–Local	2	1	500	0.0001
Local–Regional	150	30	50	0.015
Local–Global	500	150	10	0.03
Regional–Regional	40	10	100	0.005
Regional–Global	350	60	20	0.008
Global–Global	60	15	200	0.001

C.3 Real-World Azure Setting Details.

This section describes the real-world deployment used for the Azure-based HACO experiments. Unlike the simulated experiments in Appendix C.2.1, this setting does not sample latency from a hand-specified topology. Instead, each candidate agent is backed by an Azure OpenAI deployment in a concrete cloud region, and the runtime measures endpoint round-trip time during execution.

Azure candidate pool. The deployment uses the same four MAS roles as the main experiments: planner, coder, debugger, and filter. For each role, we instantiate one candidate in each of four Azure regions, yielding 16 role candidates in total. The router/evaluator is a separate Japan East deployment of `gpt-4.1-mini` and is not counted as a role candidate. In real mode, the configuration loader canonicalizes the Azure connection fields and dispatches these entries through the `azure-openai-real-chat` backend. Table 7 summarizes the active model and region assignments; API keys and endpoint credentials are omitted from the paper.

Table 7: Azure-backed candidate pool used in the real-world bad-agent setting. The Local/Regional/Global labels are experimental execution labels; the actual network path is measured against the concrete Azure endpoint for each candidate.

Execution label	Azure region	Model	Roles instantiated	Forced outage in stress config
Global	East US	`gpt-5.1-chat`	Planner, coder, debugger, filter	Planner, coder, and filter use `real_failure_rate=1.0`
Global	Sweden Central	`gpt-4o`	Planner, coder, debugger, filter	Debugger uses `real_failure_rate=1.0`
Regional	Southeast Asia	`gpt-4.1-nano`	Planner, coder, debugger, filter	None
Local	Japan East	`gpt-4.1-mini`	Planner, coder, debugger, filter; router/evaluator	None

All deployments use Azure API version `2024-12-01-preview`, `top_p=1`, and frequency penalty 0. The planner, debugger, filter, and router/evaluator are run with temperature 0.0, while the coder role uses temperature 0.2. The bad-agent configuration injects failures only through `real_failure_rate` in real mode; candidates with value 1.0 fail before the model request is issued and are logged as agent failures, while candidates with value 0 remain available unless the actual Azure request fails.

Real network-latency measurement. For each real candidate invocation, the directly observed API latency is an end-to-end latency: it includes client serialization, Internet and cloud routing, Azure front-end handling, service queueing, model inference, response transfer, and client-side response processing. Azure does not expose a per-request decomposition that cleanly separates pure network delay from model-side processing time. We therefore log this observed full-call latency as the real candidate execution latency, while collecting an additional RTT signal to build the network profile used by HACO.

Concretely, the runtime performs a lightweight sidecar probe to the target candidate's Azure endpoint by issuing HTTP requests to the Azure deployment-listing endpoint for that resource. For each probe, it measures wall-clock round-trip time with a monotonic timer. The default probe configuration uses three samples, a 5 second per-request timeout, a 30 second cache TTL per endpoint and API version, and a 0.8 second fallback latency if every probe attempt fails. Any HTTP response is sufficient for measuring RTT, because the probe is used only to characterize transport reachability and endpoint responsiveness, not generation quality.

The median successful probe time is recorded as the endpoint RTT observation for that candidate's network profile. The runtime also serializes the transmitted notebook cells to JSON and records the UTF-8 byte size as the payload size. In real-probe mode, the source and target execution-zone labels are retained for reporting, but the network profile is endpoint-based: it measures the orchestrator-to-Azure round-trip time to the target candidate's region, rather than a synthetic zone-to-zone delay. The experience log aggregates these RTT observations into average RTT, RTT jitter, and probe failure rate for each observed agent pair. During HACO selection, the router uses the average RTT when available, or performs a fresh endpoint probe for cold-start candidates. This RTT profile serves as the real-deployment analogue of the communication-latency term in the simulated setting: it is combined with the candidate's historical execution latency and then passed through the same bounded logarithmic latency discount used in the main HACO objective.

C.4 Measured Azure Trace and Live/Replay Evaluation

Measured service behavior. We collected 300 synchronized rounds over four Azure OpenAI endpoints from 2026-07-25 15:52:42 to 18:22:38 UTC, yielding 1,200 calls with SDK retries disabled. There were 995 successes (82.92%), 119 timeouts, and 86 connection errors. Endpoint success rates were 80.67–86.00%, median end-to-end API latency 3.98–5.49s, and p95 latency 10.25–14.90s. These are end-to-end API measurements that include service and model processing; they differ from the RTT probes described above.

Protocol and scope. The Azure evaluation uses 30 MatplotBench tasks and 24 logical candidates (six per planner, coder, debugger, and filter role) across four physical Azure failure domains. Tasks 1–15 use live observations. For Tasks 16–30, East US, Sweden Central, and Japan East receive the recorded success, timeout, connection-error and latency states in chronological order, with replay latency added to live call latency; Southeast Asia remains a live-only healthy control. Candidates sharing a deployment share its replayed state. HACO and UCB-Single start from identical uninformative priors and maintain separate execution histories. The deterministic outage in Fig. 3 provides a complementary controlled stress test.

Phase	Method	Effectiveness		Efficiency		Reliability
		Avg. Score	SR@80 (%)	Tokens (k)	Latency (s)	All-Failed count/rate
All 30	HACO	**65.00**	**46.67**	80.08	19.33	3/160 (1.88%)
	UCB-Single	55.67	23.33	**68.82**	**14.27**	1/405 (0.25%)
Tasks 1–15 (live)	HACO	**68.67**	**40.00**	94.31	18.42	0/84 (0.00%)
	UCB-Single	57.67	20.00	**68.66**	**17.02**	0/197 (0.00%)
Tasks 16–30 (replay)	HACO	**61.33**	**53.33**	**65.84**	20.24	3/76 (3.95%)
	UCB-Single	53.67	26.67	68.98	**11.52**	1/208 (0.48%)

Table 8: Azure live/replay results. Effectiveness follows the definitions in Table 12. SR@80 is strict success. Tokens (k) is mean total token count and Latency (s) is mean agent latency over all tasks in each phase. All-Failed is computed over all observed execution steps in each phase and is shown as count/denominator (rate). Execution-step counts and trace exposure are policy-dependent.

Quality, exposure, and recovery. Table 8 shows a 9.33-point mean-score gain over all 30 tasks. During replay HACO scores 61.33 versus 53.67. Of its 192 candidate attempts, 19 fail; six of the nine role invocations containing a failed attempt retain a successful alternative. Three of 76 role invocations are all-failed, but all 15 tasks produce outputs. UCB-Single has one all-failed role invocation out of 208.

From live to replay, HACO's mean hedge size changes from 3.06 to 2.53 and its healthy-control share from 40.86% to 46.88%; UCB-Single's healthy-control share rises from 83.76% to 97.61%. HACO retains cross-domain diversity, whereas UCB-Single concentrates on the healthy endpoint. The live and replay phases use different task subsets.

C.5 Compute Resources.

All experiments were executed in Dockerized environments (Docker 29.1.3) on a Linux server with AMD EPYC 9J14 CPUs (2 sockets, 96 cores per socket, 384 logical CPUs), 755 GiB system memory, and NVMe storage (6.9 TB total). Our method does not require local GPU training/inference; all model inference is performed via external LLM APIs, and the same API/deployment settings are used across compared methods for fairness. Although the host machine has GPUs available, they were not used in the reported experiments. Detailed runtime/token/cost statistics are already reported in Appendix Table 12.

D Algorithms and Baselines

D.1 HACO Algorithm

Algorithm 1 summarizes HACO. Here, τ specifies the target system reliability, λ controls the exploration bonus in optimistic ranking, and γ controls the conservative margin used in hedge-set stopping. The controller mapping Φ takes the current query-context pair $(q, \mathcal{C}_t)$ and returns a role invocation event (r_t, x_t). The initial task context $\mathcal{C}_0$ is initialized from the original query q, input data, and an empty execution history.

Hyperparameters. Unless otherwise specified, all HACO results use $\tau = 0.9$, $\lambda = 1.1$, $\gamma = 0.7$, $\kappa = 2.0$, $\eta = 0.5$, $\epsilon_{\mathrm{B}} = 10^{-6}$, and $\epsilon_{\mathrm{R}} = 0.1$. Here, τ is the target system reliability, λ is the optimistic-ranking coefficient, γ is the conservative-margin coefficient, κ is the bandwidth-jitter penalty, η is the latency-discount strength, ϵ_{B} is the bandwidth denominator floor, and ϵ_{R} is the floor on the capability term before weighting by physical reliability. Only τ is varied in the reliability-target sweep. For the BestOne baseline, we use cold-start threshold $n_{\min} = 2$ and epsilon-greedy exploration probability $p_{\exp} = 0.1$.

D.2 Properties of HACO's Redundancy Allocation

Scope of the guarantee. The results in this section should be interpreted as conditional properties of the HACO allocation rule, with their validity depending on explicit modeling assumptions. In particular, the conservative success statement relies on two conditions: the candidate qualified-success estimates R_i^- are calibrated lower bounds on the corresponding true probabilities, and selected candidates are conditionally independent given the observed runtime state. Under these assumptions, HACO's accumulation rule provides a conservative certificate for the selected hedge set. When the assumptions are violated, the certificate should be viewed as an approximate reliability estimate, and empirical monitoring or recalibration is required.

We provide several basic properties of HACO's Phase-2 redundancy allocation rule. Recall that candidates are sorted by descending optimism-adjusted utility U_i^+, and then added sequentially using the conservative execution reliability estimate $R_i^- \in [0, 0.98]$, with the cumulative failure estimate updated as $P_{\mathrm{fail}} \leftarrow P_{\mathrm{fail}}(1 - R_i^-)$. The algorithm stops once $1 - P_{\mathrm{fail}} \geq \tau$.

Notation. Let $m = |\mathcal{A}|$ denote the number of candidates. The sorted candidate list is $a_{(1)}, a_{(2)}, \ldots, a_{(m)}$, where $U_{(1)}^+ \geq U_{(2)}^+ \geq \cdots \geq U_{(m)}^+$. For the first k candidates in this order,

Algorithm 1: HACO: Hedged Agent Computing

Input : Query q, target system reliability τ, exploration λ, risk γ
Output : Final response

Initialize task context $\mathcal{C}_0$; // Initialize context
while *task not completed* **do**
  Phase 1: Role Abstraction
  $(r_t, x_t) \leftarrow \Phi(q, \mathcal{C}_t)$; // Infer role invocation event
  $\mathcal{A}_t \leftarrow$ candidate agent instances for r_t; // Retrieve candidates by role type
  Phase 2: Redundancy Allocation
  foreach $a_i \in \mathcal{A}_t$ **do**
    $(\mu_i, \sigma_i) \leftarrow \text{PosteriorStats}(\alpha_i, \beta_i)$; // Posterior stats
    $\rho_i \leftarrow (1 - f_{\text{int}}^{(i)})(1 - \ell_{\text{net}}^{(i)})$; // Execution success probability
    $C_i \leftarrow \ln\left(1 + \frac{T_i}{T_0}\right)$; // latency-aware resource cost
    $D_i \leftarrow 1 + \eta C_i$; // Stable delay discount
    $U_i^+ \leftarrow \frac{\min(1, \mu_i + \lambda\sigma_i)\rho_i}{D_i}$; // Optimistic ranking
  Sort $\mathcal{A}_t$ by U_i^+ descending; // Rank by optimistic utility
  $\mathcal{S}_t \leftarrow \emptyset$, $P_{\text{fail}} \leftarrow 1$; // Initialize hedge
  foreach a_i *in sorted* $\mathcal{A}_t$ **do**
    $R_i^- \leftarrow \min(0.98, \max(\epsilon_{\text{R}}, \mu_i - \gamma\sigma_i) \cdot \rho_i)$; // Conservative execution reliability
    $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup \{a_i\}$; // Add to hedge
    $P_{\text{fail}} \leftarrow P_{\text{fail}}(1 - R_i^-)$; // Update failure
    if $1 - P_{\text{fail}} \geq \tau$ **then**
      break; // Meet target system reliability
  Phase 3: Hedged Execution
  Launch $\mathcal{S}_t$ on payload x_t in parallel; // Parallel launch
  Monitor completions until a_{winner} is determined or all candidates fail; // Decision-time latency
  $\mathcal{S}_t^+ \leftarrow \{a_i \in \mathcal{S}_t \mid s_i = 1\}$; // Successful candidates
  if $\mathcal{S}_t^+ = \emptyset$ **then**
    $\mathcal{C}_{t+1} \leftarrow \text{IntegrateFailure}(\mathcal{S}_t)$; // Trigger recovery
    continue
  $a_{\text{winner}} \leftarrow \arg\max_{a_i \in \mathcal{S}_t^+} U_i^+$; // Winner selection
  $\mathcal{C}_{t+1} \leftarrow \text{Integrate}(a_{\text{winner}})$; // Context update
  Collect remaining traces asynchronously; // Experience harvesting
  Phase 4: Experience Harvesting
  foreach $a_i \in \mathcal{S}_t$ **do**
    Record execution trace; // (q_i, s_i, t_i) + telemetry
    Update candidate profile; // Candidate statistics
    Update link profile; // Link statistics
return final response

define

$$P_{\text{fail}}^{(k)} := \prod_{j=1}^{k} (1 - R_{(j)}^-), \qquad P_{\text{succ}}^{(k)} := 1 - P_{\text{fail}}^{(k)}.$$

By convention, $P_{\text{fail}}^{(0)} = 1$ and $P_{\text{succ}}^{(0)} = 0$.

Reachability assumption. Unless otherwise stated, the following guarantees are stated for the feasible case in which the target reliability is conservatively reachable by the available candidate pool:

$$P_{\text{succ}}^{(m)} = 1 - \prod_{j=1}^{m} (1 - R_{(j)}^-) \geq \tau.$$

If this condition does not hold, HACO returns the full candidate pool under the given ordering. In that case, the conservative success estimate is $P_{\text{succ}}^{(m)}$, and no method restricted to the same candidate pool and conservative estimates can certify the target τ.

Lemma D.1 (Monotonicity of conservative accumulation)**.** *Assume $R^-_{(j)} \in [0,1]$ for all j. Then $P^{(k)}_{\text{fail}}$ is nonincreasing in k, and $P^{(k)}_{\text{succ}}$ is nondecreasing in k.*

Proof. Since $R^-_{(k+1)} \in [0,1]$, we have $1 - R^-_{(k+1)} \in [0,1]$. Therefore

$$P^{(k+1)}_{\text{fail}} = P^{(k)}_{\text{fail}}(1 - R^-_{(k+1)}) \le P^{(k)}_{\text{fail}}.$$

Hence $P^{(k)}_{\text{fail}}$ is nonincreasing, and thus $P^{(k)}_{\text{succ}} = 1 - P^{(k)}_{\text{fail}}$ is nondecreasing. □

Proposition D.2 (Conditional conservative qualified-success certificate)**.** *Assume conditional independence across selected candidates, suppose that R^-_i is a lower bound on the true qualified-success probability of candidate a_i, and assume that the target reliability is conservatively reachable, i.e., $P^{(m)}_{\text{succ}} \ge \tau$. If HACO terminates with hedge set $\mathcal{S}$, then*

$$\mathbb{P}(\textit{qualified success} \mid \mathcal{S}) \ge \tau.$$

Proof. Under conditional independence, the true probability that no selected candidate achieves qualified success is

$$\prod_{a_i \in \mathcal{S}} (1 - p_i),$$

where p_i denotes the true qualified-success probability of candidate a_i. Since $R^-_i \le p_i$, we have

$$1 - p_i \le 1 - R^-_i,$$

and therefore

$$\prod_{a_i \in \mathcal{S}} (1 - p_i) \le \prod_{a_i \in \mathcal{S}} (1 - R^-_i) = P_{\text{fail}}.$$

Thus,

$$\mathbb{P}(\text{qualified success} \mid \mathcal{S}) = 1 - \prod_{a_i \in \mathcal{S}} (1 - p_i) \ge 1 - P_{\text{fail}}.$$

By the reachability assumption and the stopping rule, HACO stops at some $k^\star \le m$ satisfying $1 - P_{\text{fail}} \ge \tau$. Therefore the result follows. □

Proposition D.3 (Minimal feasible prefix under HACO ordering)**.** *Fix the ordering induced by descending U^+_i, and assume $P^{(m)}_{\text{succ}} \ge \tau$. Suppose HACO stops after selecting the first $k^\star$ candidates. Then the returned hedge set is exactly the shortest prefix of this ordered list whose conservative success estimate reaches the target:*

$$P^{(k^\star)}_{\text{succ}} \ge \tau, \qquad P^{(k)}_{\text{succ}} < \tau \quad \textit{for all } k < k^\star.$$

Proof. By construction, HACO scans the ordered list sequentially and stops at the first index $k^\star$ such that $P^{(k^\star)}_{\text{succ}} \ge \tau$. Therefore all earlier prefixes fail to meet the threshold, i.e., $P^{(k)}_{\text{succ}} < \tau$ for $k < k^\star$. Hence the selected set is the shortest feasible prefix under the HACO ordering. □

Proposition D.4 (Monotonicity with respect to the system reliability target)**.** *Fix the ordered candidate list and the conservative execution reliability estimates $\{R^-_{(j)}\}_{j=1}^m$, and assume $P^{(m)}_{\text{succ}} \ge \tau_2$. Let $k^\star(\tau)$ denote the stopping index of HACO under target system reliability τ. Then $k^\star(\tau)$ is nondecreasing in τ: if $\tau_1 \le \tau_2$, then*

$$k^\star(\tau_1) \le k^\star(\tau_2).$$

Proof. From Lemma D.1, $P^{(k)}_{\text{succ}}$ is nondecreasing in k. For a lower threshold τ_1, HACO stops at the smallest prefix whose system reliability estimate reaches τ_1. For a higher threshold $\tau_2 \ge \tau_1$, any prefix feasible for τ_2 is also feasible for τ_1, but not necessarily conversely. Therefore the minimal feasible prefix for τ_2 cannot be shorter than that for τ_1. □

Proposition D.5 (Single-step overshoot bound)**.** *Assume $P_{\mathrm{succ}}^{(m)} \geq \tau$, and let $k^\star$ be the stopping index. Then HACO's conservative success estimate satisfies*

$$0 \leq P_{\mathrm{succ}}^{(k^\star)} - \tau \leq P_{\mathrm{fail}}^{(k^\star-1)} R_{(k^\star)}^{-}.$$

Proof. We have

$$P_{\mathrm{succ}}^{(k^\star)} - P_{\mathrm{succ}}^{(k^\star-1)} = P_{\mathrm{fail}}^{(k^\star-1)} R_{(k^\star)}^{-}.$$

Since $k^\star$ is the first index reaching the threshold,

$$P_{\mathrm{succ}}^{(k^\star-1)} < \tau \leq P_{\mathrm{succ}}^{(k^\star)}.$$

Hence

$$0 \leq P_{\mathrm{succ}}^{(k^\star)} - \tau \leq P_{\mathrm{succ}}^{(k^\star)} - P_{\mathrm{succ}}^{(k^\star-1)} = P_{\mathrm{fail}}^{(k^\star-1)} R_{(k^\star)}^{-}.$$

□

D.3 Correlated Failure Domains and Common Shocks

The original product certificate is a special case of a domain-level bound. Let x denote the pre-execution state (including allocation history), let $\mathcal{S}$ be a set selected using that state, and let E_i denote candidate i producing a physically successful, quality-qualified output. Partition candidates into M failure domains $G_1, \ldots, G_M$ according to shared infrastructure. Let H be a common-shock event and assume $0 \leq \bar{q}(x) \leq 1$ with $\Pr(H \mid x) \leq \bar{q}(x)$. For $\Pr(H^c \mid x) > 0$, suppose $\underline{p}_i^{(0)}(x) \leq \Pr(E_i \mid H^c, x)$ are calibrated lower bounds.

Proposition D.6 (Domain-level conditional certificate)**.** *Suppose the domain-success events $A_k = \bigcup_{i \in \mathcal{S} \cap G_k} E_i$ are independent conditional on (H^c, x), allowing arbitrary dependence within a domain. Set $b_k(\mathcal{S}, x) = \max_{i \in \mathcal{S} \cap G_k} \underline{p}_i^{(0)}(x)$, with $\max \emptyset = 0$. Then*

$$\Pr\left(\bigcup_{i \in \mathcal{S}} E_i \mid x\right) \geq L(\mathcal{S}, x) := (1 - \bar{q}(x)) \left[1 - \prod_{k=1}^{M} (1 - b_k(\mathcal{S}, x))\right]. \tag{23}$$

Proof. For each domain, $\Pr(A_k \mid H^c, x) \geq b_k(\mathcal{S}, x)$. Conditional independence gives $\Pr(\bigcup_k A_k \mid H^c, x) \geq 1 - \prod_k (1 - b_k)$. Multiplying by $\Pr(H^c \mid x) \geq 1 - \bar{q}(x)$ and discarding the nonnegative success contribution conditional on H proves the claim. If $\Pr(H^c \mid x) = 0$, necessarily $\bar{q} = 1$ and the trivial lower bound is zero. □

For $\bar{q} = 0$ and singleton domains this reduces to the original independent-candidate certificate. Let $\underline{p}_i(x) \leq \Pr(E_i \mid x)$ be unconditional marginal lower bounds. Without cross-domain independence, they give the dependence-agnostic certificate $\Pr(\bigcup_{i \in \mathcal{S}} E_i \mid x) \geq \max_{i \in \mathcal{S}} \underline{p}_i(x)$; using conditional bounds instead gives $(1 - \bar{q}) \max_{i \in \mathcal{S}} \underline{p}_i^{(0)}(x)$. The infrastructure incidents discussed in the introduction motivate this shared-failure model.

The bound in Eq. (23) certifies the existence of a qualified candidate in $\mathcal{S}$. It is bounded above by $1 - \bar{q}$, so targets above this value cannot be certified by this bound. If $q_{\mathrm{sh}} = \Pr(H \mid x)$ and success is impossible during a shock, the true success probability is at most $1 - q_{\mathrm{sh}}$.

The experiments use the original product accumulator with R_i^- as an operational stopping score; Eq. (23) is a theoretical extension. Appendix E.2 evaluates aggregate conservativeness and target reachability, while the shared-state Azure replay evaluates adaptation under correlated failures.

D.4 Baselines

We compare our method with four routing baselines: RANDOM, BESTONE, UCB-SINGLE, and MOA-STYLE BEST-OF-N.

RANDOM. This baseline uniformly samples one candidate from the candidate pool and executes only that candidate. Its output is returned directly without any further reranking or model-based evaluation.

Algorithm 2: RANDOM

Input: Query x, candidate set $\mathcal{A}$
Output: Final response y

Uniformly sample one candidate a from $\mathcal{A}$;
Execute candidate a on query x and obtain response y;
return y;

Algorithm 3: BESTONE

Input: Query x, candidate set $\mathcal{A}$, exploration probability p_{exp}
Output: Final response y

foreach *candidate* $a \in \mathcal{A}$ **do**
    Estimate its historical average utility $u(a)$;
    if *candidate* a *has no history* **then**
        Assign an optimistic default utility;
Let $a^\star$ be the candidate with the highest estimated utility;
Draw a random number r from $[0, 1]$;
if $r > p_{\text{exp}}$ **then**
    Select $a^\star$;
else
    Randomly select one candidate from $\mathcal{A} \setminus \{a^\star\}$;
Execute the selected candidate on query x and obtain response y;
return y;

BESTONE. This baseline performs single-candidate selection based on historical performance. It uses the same LLM-judge quality signal as HACO for executed candidates, with failed or invalid executions assigned zero utility. It prefers the candidate with the best past average utility, while retaining a small probability of selecting other candidates for exploration. Only the selected candidate is executed and updated, and its response is returned as the final output.

MoA-style Best-of-N. This baseline executes all candidates in parallel and collects all generated responses. A separate evaluator language model then assigns output-quality scores to the candidate outputs, and the highest-scoring response is selected as the final answer. We use "MoA" as a compact figure and table label for this MoA-style full-pool evaluator-based baseline.

UCB-Single. UCB-Single ranks candidates using the same U_i^+, capability and environment/link profiles, latency discount, and update rules as HACO, but executes only the top-ranked candidate. It starts from the same uninformative priors and maintains its own experience log. HACO also harvests feedback from its parallel branches for subsequent routing decisions.

Discussion. These baselines represent four distinct routing paradigms: random single-candidate routing, experience-guided single-candidate routing with limited exploration, uncertainty-aware top-1 routing, and parallel candidate generation followed by model-based selection. They provide simple reference points for evaluating the effectiveness-efficiency trade-off of routing strategies.

E More Experimental Details

E.1 Radar Axis Definitions

For each benchmark b and method a, the radar in Fig. 4 is built from five raw metrics and then mapped to a common higher-is-better radial scale.

Effectiveness and strict success. Let $\mathcal{T}_{a,b}$ be tasks of method a on benchmark b. The first axis ("Avg. Score" or "Accuracy") is the benchmark-specific mean effectiveness in percentage:

$$M_{a,b}^{\text{eff}} = \begin{cases} \frac{1}{|\mathcal{T}_{a,b}|} \sum_{t \in \mathcal{T}_{a,b}} \text{score}_t, & b \in \{\text{MatplotBench}, \text{DSBench}\}, \\ \frac{1}{|\mathcal{T}_{a,b}|} \sum_{t \in \mathcal{T}_{a,b}} \text{acc}_t, & b = \text{InfiAgent-Bench}. \end{cases}$$

Algorithm 4: ALGORITHM 4: MOA-STYLE BEST-OF-N ROUTER

Input: Query x, candidate set $\mathcal{A}$
Output: Final response y

Execute all candidates in $\mathcal{A}$ on query x in parallel;
Collect all candidate responses $\{y_a \mid a \in \mathcal{A}\}$;
Use an evaluator language model to assign an output-quality score to each candidate response;
Select the highest-scoring response as y;
return y;

Here, score_t is the benchmark task score and acc_t is proportional sub-question accuracy, both on a percentage scale. The strict-success axis, with $\mathbf{1}[\cdot]$ denoting the indicator function, is:

$$M_{a,b}^{\text{strict}} = \begin{cases} 100 \cdot \frac{1}{|\mathcal{T}_{a,b}|} \sum_t \mathbf{1}[\text{score}_t \geq 80], & b = \text{MatplotBench}, \\ 100 \cdot \text{question-level success fraction}, & b = \text{InfiAgent-Bench}, \\ 100 \cdot \frac{\#\{t:\text{score}_t > 0\}}{|\mathcal{T}_{a,b}|}, & b = \text{DSBench}. \end{cases}$$

Raw cost and failure metrics. Token and latency columns report per-task means from the corresponding benchmark summaries. Let $\mathcal{C}_{a,b}$ denote the task set used for these cost aggregates, and $\mathcal{T}_{a,b}$ the task set used for effectiveness. LLM-judge tokens for q_i are counted, while asynchronous judging does not add to blocking latency:

$$M_{a,b}^{\text{tok}} = \frac{1}{1000} \cdot \frac{1}{|\mathcal{C}_{a,b}|} \sum_{t \in \mathcal{C}_{a,b}} \text{token_count}_t, \qquad M_{a,b}^{\text{lat}} = \frac{1}{|\mathcal{C}_{a,b}|} \sum_{t \in \mathcal{C}_{a,b}} \text{latency}_t.$$

For reliability, let $\mathcal{E}_{a,b}$ be all observed task steps and let $\mathcal{B}_{t,s}$ be executed concurrent branches at step s of task t. With branch success indicator $z_{t,s,k} \in \{0, 1\}$, the all-failed-step rate is

$$M_{a,b}^{\text{fail}} = 100 \cdot \frac{1}{|\mathcal{E}_{a,b}|} \sum_{(t,s) \in \mathcal{E}_{a,b}} \mathbf{1}\left[\sum_{k \in \mathcal{B}_{t,s}} z_{t,s,k} = 0\right].$$

Lower $M_{a,b}^{\text{fail}}$ means higher reliability.

"Token Eff.", "Latency Eff.", and "Reliability" in the radar. For each benchmark and each axis j, let $x_{a,b,j}$ be the raw value above, and define method-wise extrema $x_{b,j}^{\min} = \min_a x_{a,b,j}$, $x_{b,j}^{\max} = \max_a x_{a,b,j}$. We then apply the same monotonic mapping as in plotting:

$$\hat{x}_{a,b,j} = \begin{cases} \dfrac{x_{a,b,j} - x_{b,j}^{\min}}{x_{b,j}^{\max} - x_{b,j}^{\min}}, & \text{higher-is-better axes}, \\ \dfrac{x_{b,j}^{\max} - x_{a,b,j}}{x_{b,j}^{\max} - x_{b,j}^{\min}}, & \text{lower-is-better axes}. \end{cases}$$

Thus, "Token Eff." and "Latency Eff." are reversed versions of mean tokens and mean latency, and "Reliability" is the reversed all-failed-step rate. The implementation adds a small per-axis padding before radius projection for visual spacing, but this does not change the ordering of methods.

Raw overall benchmark results. Table 12 reports the raw numerical results used to construct the radar comparison in Fig. 4. We report two effectiveness metrics for each benchmark: Avg. Score / Acc. and Strict Succ. (%). Because the benchmarks use different evaluation protocols, the exact definitions are benchmark-specific.

For MatplotBench, Avg. Score / Acc. is the mean evaluator score over the plotting tasks. Strict Succ. (%) is SR@80, which counts a task as successful if its evaluator score is at least 80.

For InfiAgent-Bench, Avg. Score / Acc. is the proportional sub-question accuracy using all benchmark questions as the denominator. Each question receives partial credit according to the fraction of its sub-questions answered correctly, while unevaluated questions are counted as zero. Strict Succ. (%) is the question-level success rate using all benchmark questions as the denominator; a question is counted as successful only when all of its sub-questions are answered correctly.

For DSBench, Avg. Score / Acc. is the average normalized task score over all tasks, where task performance is normalized relative to the baseline and ground-truth performance. Tasks with missing or invalid outputs are counted as zero. Strict Succ. (%) is the percentage of tasks with a positive normalized score, meaning the method improves over the baseline on that task.

These benchmark-specific definitions also explain why the relative magnitude of Avg. Score / Acc. and Strict Succ. (%) differs across benchmarks. On InfiAgent-Bench, partial sub-question credit can make Avg. Score / Acc. higher than Strict Succ. On DSBench, many tasks may obtain positive but small normalized scores, so Strict Succ. can be higher than Avg. Score / Acc. On MatplotBench, the relation depends on the distribution of evaluator scores around the SR@80 threshold. Therefore, the two columns should be interpreted according to each benchmark's metric definition rather than compared using a universal ordering.

E.2 Aggregate Conservativeness and Target Reachability

For the operating points of Fig. 6, empirical qualified success means at least one physically successful candidate receiving a router-judge score of at least 8 on a 0–10 scale. We reconstruct the operational set estimate at each routing decision and compare its mean with the observed qualified-success fraction in Table 9. The estimate is conservative in aggregate at these four points. As τ rises from 0.70 to 0.95, mean hedge size increases from 2.06 to 3.06, the target-reached fraction decreases from 96.76% to 86.41%, and pool exhaustion rises from 3.24% to 13.59%. Pool exhaustion records decisions that terminate before the operational estimate reaches τ.

Table 9: Aggregate qualified-success diagnostic; rates and estimates are percentages.

Target τ	0.70	0.80	0.90	0.95
Empirical qualified success	91.82	93.97	95.89	96.81
Mean operational set estimate	87.49	93.27	92.48	95.54

E.3 Controlled Ablations Across Benchmarks

The ablations in Table 10 keep the candidate pool, models, non-target components and hyperparameters fixed. Candidate-specific profiling is replaced by a role-shared profile while retaining feedback collection. Removing the environment/link profile excludes physical-success and link/latency information from allocation while preserving candidate capability profiles. Mean-only ranking removes the optimistic bonus. The w/o-LCB variant replaces the conservative score by the posterior mean but retains adaptive accumulation; Fixed-K selects the top three candidates without threshold stopping. UCB-Single executes only the top-ranked candidate.

All six variants lower mean effectiveness on both benchmarks. Candidate-specific attribution has the largest observed score loss. On MatplotBench, Fixed-K lowers score from 58.86 to 52.39 while increasing recorded token cost from 65.32k to 82.64k and latency from 23.50s to 35.95s, despite a lower all-failed-step rate. The w/o-LCB comparison tests conservative estimation, whereas Fixed-K tests adaptive stopping.

E.4 Note on MoA-style Baseline.

MoA-style baseline's final score depends on post-hoc evaluator selection over heterogeneous candidate outputs. On InfiAgent-Bench, tasks are relatively easy and candidate outputs are often close, so full-pool execution brings limited marginal gain. On DSBench, tasks are more difficult and the pool may not contain a clearly superior output. On MatplotBench, diverse visualization programs and layouts further increase evaluator-selection noise, so MoA improves step-level availability but does not necessarily achieve the best task-level score.

Selection diagnostic. MoA executes the full pool and returns the evaluator-selected output. Across the logged settings, at least two candidates tie for the top evaluator score in 88.9–95.3% of decisions, all candidates tie in 31.8–48.6%, and the deterministic tie rule selects the first candidate in 73.7–80.1%. This reveals order-sensitive selection under limited score discrimination.

Table 10: Cross-benchmark component ablation results. Effectiveness definitions follow Table 12; cost columns retain the recorded evaluation aggregates discussed in Appendix E.1. Strict Succ. denotes SR@80 on MatplotBench and question-level accuracy on InfiAgent-Bench. All-failed step rate is the proportion of observed task steps for which every executed concurrent branch fails. Dashes indicate metrics not reported in the MatplotBench w/o-LCB summary.

Dataset	Variant	Effectiveness		Efficiency		Reliability
		Avg. Score / Acc.	Strict Succ. (%)	Tokens (k)	Latency (s)	All-Failed Step Rate (%)
MatplotBench	Full HACO	58.86	36.00	65.32	23.50	0.17
	w/o Candidate-Specific Profile	38.25	22.00	148.74	54.83	0.00
	w/o Environment/Link Profile	53.00	34.00	57.77	20.12	0.17
	w/o Optimistic Prioritization	51.61	34.00	71.55	21.83	0.00
	w/o LCB	53.90	36.00	–	–	–
	Fixed-K Allocation	52.39	30.00	82.64	35.95	0.00
	UCB-Single	51.66	32.00	42.18	46.11	1.52
InfiAgent-Bench	Full HACO	89.01	84.44	91.04	31.31	0.37
	w/o Candidate-Specific Profile	80.65	75.88	116.73	40.61	0.09
	w/o Environment/Link Profile	87.74	83.66	57.97	35.84	0.62
	w/o Optimistic Prioritization	85.93	82.10	139.84	55.11	0.21
	w/o LCB	83.04	77.43	133.48	54.79	0.26
	Fixed-K Allocation	85.86	80.93	72.24	29.46	0.07
	UCB-Single	80.35	75.88	55.14	63.70	0.29

Post-hoc winner zones. MoA always executes the full pool across zones. Table 11 therefore describes the region of the evaluator-selected output, not a pre-execution routing policy.

Condition	First 50 L/R/G (%)	Last 50 L/R/G (%)	Δ L/R/G (pp)	Calls F/L
Network degradation	16.81/40.74/42.45	39.56/60.44/0.00	+22.75/+19.70/−42.45	351/503
Agent degradation	16.81/40.74/42.45	32.89/48.29/18.82	+16.08/+7.55/−23.63	351/526
Evaluator degradation	16.81/40.74/42.45	30.20/31.10/38.70	+13.39/−9.64/−3.75	351/447
Real-agent degradation	24.27/35.67/40.06	37.09/61.39/1.52	+12.82/+25.72/−38.54	342/461

Table 11: MoA evaluator-selected winner-zone distributions for the four conditions in Fig. 3(d). L/R/G denote Local, Regional, and Global; Δ is Last 50 minus First 50 in percentage points. Calls are the numbers of valid router decisions in the two splits. These distributions describe post-hoc winner selection; MoA executes the full candidate pool across all zones.

Fig. 10 further analyzes these raw results by comparing the distribution of latency and token usage, together with task effectiveness and step-level failure rate, for the original baseline set and HACO. The added UCB-Single comparison is reported separately in Tables 1 and 12.

E.5 Note on fixed-global in InfiAgent-Bench.

The low InfiAgent-Bench score of the fixed-global baseline mainly reflects its low valid-output completion rate, while completed outputs remain of reasonable quality. When fixed-global produces a valid final output, the judge usually assigns a full score, but many runs fail to produce a valid evaluable answer and are counted as zero in the aggregate accuracy. The reported all-failed-step rate is a step-level execution-availability metric and does not directly measure task-level valid-output completion. Thus, the anomalously low average score reflects completion failures under the fixed global route, not a systematic quality drop of the underlying GPT-4o model.

E.6 Execution Environment-Adjusted Reference to External Baselines on MatplotBench

To provide an external reference, we compare HACO with strong results reported in the original DATAWISE paper [54] based on MatplotBench. Those results were obtained under the original benchmark setting, which does not model the practical heterogeneous runtime conditions considered in this work. We therefore report them as *idealized execution environment-free* scores and provide an *execution environment-adjusted reference* under our execution setting.

Table 12: Detailed information about overall benchmark performance results. Effectiveness metrics are dataset-specific. "Strict Succ." denotes SR@80 on MatplotBench and question-level accuracy on InfiAgent-Bench. "All-Failed Step Rate" is computed as the proportion of observed task steps for which all executed concurrent branches fail.

Dataset	**Method**	**Effectiveness**		**Efficiency**		**Reliability**
		Avg. Score / Acc.	Strict Succ. (%)	Tokens (k)	Latency (s)	All-Failed Step Rate (%)
MatplotBench	random	46.45	27.00	46.51	82.69	8.78
	bestone	47.75	25.00	32.22	52.50	9.35
	moa	42.75	23.00	305.67	206.41	0.00
	fix-local	23.90	11.00	43.55	54.42	0.28
	fix-regional	47.68	25.00	64.91	50.74	3.01
	fix-global	52.61	36.00	16.29	28.62	8.25
	UCB-Single	51.66	32.00	42.18	46.11	1.52
	haco	58.86	36.00	65.32	23.50	0.17
InfiAgent-Bench	random	86.06	83.27	25.82	34.87	8.65
	bestone	77.82	75.49	25.33	22.69	13.94
	moa	84.79	80.93	204.02	99.82	0.00
	fix-local	69.42	64.98	52.37	132.47	0.13
	fix-regional	87.86	84.44	35.81	25.82	3.56
	fix-global	4.28	4.28	14.27	13.23	8.59
	UCB-Single	80.35	75.88	55.14	63.70	0.29
	haco	89.01	84.44	91.04	31.31	0.37
DSBench	random	31.83	58.90	89.54	254.04	6.95
	bestone	41.84	73.61	98.90	107.41	11.14
	moa	41.09	71.23	711.73	423.41	0.11
	fix-local	4.90	10.96	103.52	546.77	0.00
	fix-regional	39.79	67.12	88.39	143.46	3.53
	fix-global	33.02	61.64	60.28	75.21	7.64
	UCB-Single	33.28	75.34	112.63	270.35	0.00
	haco	42.40	76.71	393.72	125.91	0.40

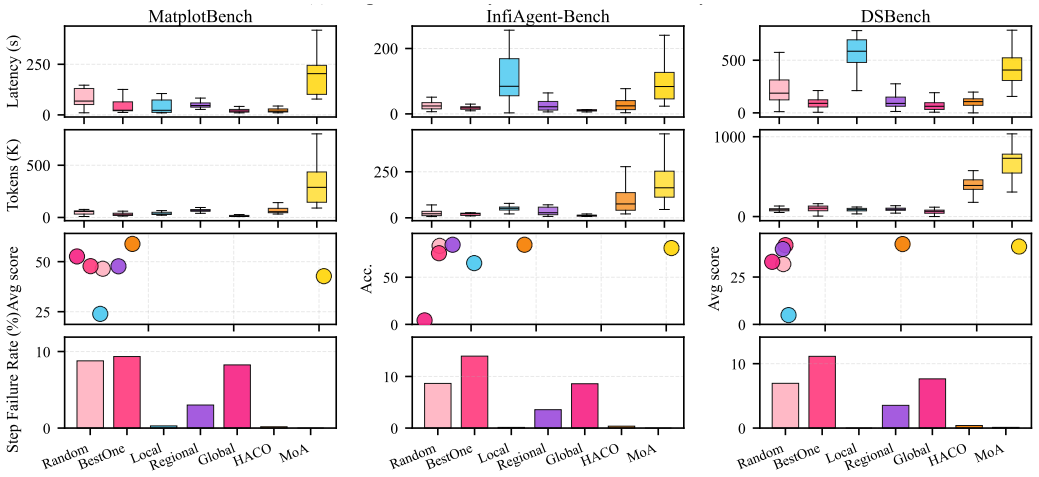


Figure 10: **More analysis of raw benchmark results.** The figure compares latency, token usage, task effectiveness, and step-level failure rate for baselines and HACO across MatplotBench, InfiAgent-Bench, and DSBench.

The adjustment is calibrated using two measured anchor points. The first anchor is the DATAWISE w/ visual tool score reported in [54]. The second anchor is the best fixed-zone MatplotBench score measured in our heterogeneous execution setting, where the same type of agent workflow is affected by the runtime conditions used in our experiments. We apply this calibration to the external methods to approximate the performance decrease caused by our network-aware execution setting. The HACO row is directly observed in our experiments. Thus, Table 13 provides a contextual comparison that approximates how real-world execution environment factors may affect results originally reported under idealized execution environment-free benchmark settings.

Table 13: External reference on MatplotBench under idealized execution environment-free and execution environment-adjusted settings. The idealized scores are taken from DATAWISE/DatawiseAgent [54]; the adjusted scores are estimated from those reported results under our heterogeneous execution-environment setting. For DATAWISE, we also report the result after adding HACO under the same adjusted setting.

Method / Reference	Idealized execution environment-free setting	Execution environment-adjusted setting	
DATAWISE w/ visual tool [54]	**64.33**	52.61 (w/o HACO)	**58.86** (w/ HACO)
AutoGen w/ visual tool [50]	63.60	52.0	
DATAWISE [54]	61.22	50.1	
MatplotAgent [53]	57.86	47.3	
Direct Decoding [54]	45.28	37.0	

F Limitations

Task scope. Our validation instantiates HACO on a data-analysis-oriented agent workflow and evaluates it on benchmarks covering data science, structured data analysis, and scientific visualization. These workflows provide controlled long-horizon testbeds with code execution, failure recovery, and measurable quality signals. However, HACO is a general runtime allocation layer that can be applied to any agent-based system exposing role invocation events and candidate agent instances. Future work should validate HACO on broader workflows, such as web automation, retrieval-intensive agents, and distributed intelligent systems, where the execution-environment factors may become more pronounced because communication links among edge devices can fluctuate more strongly.

Reliability-target tuning. HACO treats the target reliability τ as an explicit operating parameter for the reliability–cost trade-off. In practice, τ should be tuned according to application requirements, budget, and latency constraints. Future work can further study automatic target selection and adaptive reliability scheduling across tasks with different risk levels.

Feedback signals for experience harvesting. The current experience harvesting module uses operational feedback such as quality, success, latency, token usage, and network statistics. Future work can incorporate richer semantic feedback, including error types, reasoning traces, code-level failure patterns, and evaluator rationales. In addition, we plan to introduce temporal adaptation mechanisms, such as forgetting factors, temporal discounting, sliding-window updates, or drift detection, so that the Beta-based capability profiles can respond more quickly to recent observations and better track non-stationary candidate behavior.

G Broader Impacts

This work studies reliability control for LLM-based agent systems under heterogeneous runtime conditions. Positive impacts are that more reliable role invocation can reduce failures in long-horizon workflows, improve service continuity under regional or network degradation, and lower unnecessary token and latency costs compared with exhaustive parallel execution. These properties can benefit practical agent deployments in domains such as data analysis, engineering assistance, and scientific workflows, where unstable execution can otherwise waste resources or interrupt user-facing services.

Potential negative impacts should also be considered. Improving execution reliability may make agent systems easier to deploy at scale, which can amplify harms already associated with LLM systems, such as broader resource consumption from increased inference activity. In addition, system-level robustness does not guarantee factual correctness, fairness, privacy preservation, or resistance to malicious use, so more reliable execution could still propagate harmful outputs if underlying models or tools fail in these dimensions. We therefore view HACO as a systems mechanism. It could be deployed with application-level safeguards such as human oversight in high-stakes settings, usage restrictions consistent with provider policies, and monitoring of cost, latency, and misuse patterns.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction present HACO as a runtime reliability method for role invocation under heterogeneous execution conditions, and these claims are supported by the experimental results, the theory/appendix analysis, and the dedicated "Limitations" section.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The dedicated "Limitations" section discusses task scope, operating-parameter dependence (τ), and feedback-signal constraints.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper states the HACO allocation rule in "The HACO Paradigm" and provides the formal analysis in Appendix "Properties of HACO's Redundancy Allocation," including reachability, the calibrated-bound and independence assumptions, and the domain/common-shock extension (Appendix D.3).

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper and appendices disclose the benchmark/task setups, MAS settings, candidate pools, network simulation, real-world deployment configuration, baselines, and key hyperparameters, which together provide the information needed to reproduce and verify the main experimental findings to the extent relevant to the paper's claims.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is available at `https://github.com/NICE-HKU/haco.git`. The repository README provides setup and evaluation instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: "Experimental Setting" together with the appendix sections on MAS settings, candidate pools, network simulation, real-world deployment details, baselines, and algorithm/hyperparameters provides the benchmark/task setup, candidate assignments, network parameters, and key operating parameters needed to understand the reported results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports point estimates without error bars, confidence intervals, or statistical significance tests.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The appendix includes a dedicated "Compute Resources" subsection (Appendix Section C.5) that reports the execution environment (Dockerized, CPU-only runtime for reported results, and machine/worker memory-storage details), and appendix tables provide runtime/token/cost statistics; together these provide sufficient compute-resource information for reproduction.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We reviewed the NeurIPS Code of Ethics. The work is a system-level evaluation on public benchmarks and provider APIs, and it does not involve crowdsourcing, human-subject interventions, or the release of a new high-risk model or scraped dataset.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses potential positive and negative societal impacts in Appendix "Broader Impacts," including improved reliability and service continuity as benefits, and risks related to over-reliance, high-stakes misuse, and increased resource consumption, together with deployment caveats.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This work does not release a new high-risk model or scraped dataset artifact that would require a controlled-release safeguard policy in this submission.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper credits the external datasets, benchmarks, baselines, and model services through citations in the main text and appendix, and the corresponding official resources document the applicable licenses and terms of use for the assets employed in this work.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Code is available at `https://github.com/NICE-HKU/haco.git`, with documentation and instructions for setup and evaluation provided in the repository README.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing studies or experiments with human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No human-subject research is conducted in this work, so IRB-style approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM usage is central to the method and is explicitly described throughout the paper (abstract/method/experiments), with concrete model and deployment details reported in "Candidate Pool Details" and "HACO in Real-World Setting Details".

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.