BRA-AUDIT: BUDGETED RUNTIME AUDITING FOR LLM MULTI-AGENT SYSTEMS VIA CUMULATIVE-EXPOSURE AUDIT-POINT PLACEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems (LLM-MAS) rely on inter-agent dependencies that can amplify local hallucinations or malicious outputs into system-level failures. Runtime auditors can contain these failures, but auditing effectiveness depends not only on how an auditor judges a trajectory, but also on where limited audit calls are placed. End-only auditing leaves long dependency chains unchecked and enlarges rollback scope, whereas exhaustive interaction-level auditing incurs substantial token cost. We propose BRA-Audit, a budgeted runtime auditing framework that treats auditing as audit-point placement over a dynamic execution graph. Its key principle is that an audit point has non-local value: once verified, it prevents accumulated unchecked exposure from being repeatedly inherited by downstream audit contexts. BRA-Audit weights exposure by verification staleness and downstream reachability, minimizes cumulative unchecked exposure under a fixed audit-call budget, and greedily selects points by marginal exposure reduction. Verified points become trusted boundaries for localized auditing and rollback. Across AgentsNet, BBH, and MultiAgentBench-Research, BRA-Audit recovers performance close to clean execution while using 17.2%–40.6% fewer tokens than strong auditing baselines; under the same budget, its placement strategy also outperforms random auditing. Our code is available at https://anonymous.4open.science/r/BRA-7CC6/.

1 INTRODUCTION

LLM-based multi-agent systems (LLM-MAS) coordinate specialized agents through explicit roles, communication, and shared workflows Hong et al. (2024); Qian et al. (2024); Chan et al. (2024). Such systems can combine task decomposition, heterogeneous expertise, tool use, and multiple reasoning paths, enabling long-horizon and open-ended collaboration Fourney et al. (2024); Zhu et al. (2025); Wang et al. (2024). The same inter-agent dependencies, however, create a distinctive failure mode: hallucinated or malicious outputs can be reused as context Tran et al. (2025); Yan et al. (2026); Li et al. (2023), propagate through communication and aggregation, and eventually turn a local error into a system-level failure Lee & Tiwari (2024); Ju et al. (2024); He et al. (2025). As execution grows longer and more tightly coupled, the source of a failure also becomes harder to identify and contain Xia et al. (2026); Zheng et al. (2025).

Dedicated auditor agents provide a natural defense by inspecting intermediate reasoning and cross-agent interactions Xiang et al. (2024); Wang et al. (2026). Existing safeguards, however, primarily focus on *how* auditing is performed; with a limited audit budget, an equally important question is *where* limited audit calls should be placed Fan & Li (2025); Du et al. (2024). This placement problem is not simply an independent node-scoring problem. If an upstream output remains unchecked, its information can be repeatedly inherited by multiple downstream audit contexts until the dependency flow reaches a verified audit point Triedman et al. (2025); Wang et al. (2025a); Xie et al. (2025). An audit point therefore has non-local value: it both verifies the current trajectory region and prevents previously unchecked exposure from continuing into later contexts. Consequently, the value of one audit point depends on the other points already selected.

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

This coupling creates a fundamental defense–cost trade-off. End-only auditing processes a long, dependency-dense trajectory after task completion, which can obscure failure sources, delay intervention, and enlarge rollback scope Liu et al. (2024); Yang et al. (2026). At the other extreme, interaction-level auditing inspects agents frequently, enabling earlier detection and finer localization but requiring many auditor calls and substantial token cost Fan & Li (2025); Du et al. (2024). The central problem is therefore: *under a limited audit-call budget, where should audits be placed so that unchecked information does not accumulate and propagate through the MAS execution graph?*

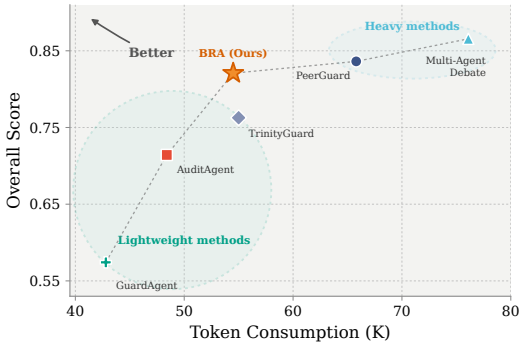


Figure 1: The performance-cost trade-off of BRA-Audit against other methods under attack in AgentsNet benchmark.

To address this problem, we propose BRA-Audit, a budget-aware runtime auditing framework that formulates audit scheduling as audit-point placement over a dynamic dependency graph. BRA-Audit assigns each runtime node a temporal–topological exposure weight: *AuditGap* captures how long its producing agent has remained unverified, while *TopoImpact* measures how broadly its output can influence downstream computation. Rather than ranking nodes independently, BRA-Audit minimizes the *cumulative unchecked exposure* carried through audit contexts under a fixed audit-call budget. Because placing one audit point changes the value of later placements, BRA-Audit greedily selects the point with the largest marginal exposure reduction and recomputes gains after each selection. A verified point then becomes a trusted boundary for subsequent execution, shortening later audit contexts and localizing rollback when an unsafe region is detected.

Experiments on AgentsNet, BIG-Bench Hard (BBH), and MultiAgentBench-Research cover structured coordination, complex multi-step reasoning, and open-ended collaboration. BRA-Audit consistently recovers task performance close to clean execution while reducing end-to-end token consumption by 17.2%–40.6% relative to strong auditing baselines such as PeerGuard and Multi-Agent Debate. Under the same audit ratio $\rho = 0.4$, cumulative-exposure placement also outperforms random placement by 0.040 on BBH and 0.081 on MultiAgentBench-Research with comparable token cost, showing that the placement policy itself contributes beyond simply increasing audit frequency.

Our contributions are as follows:

- We identify budgeted audit placement as a distinct runtime defense problem for LLM-MAS: an audit point has non-local value because it truncates unchecked exposure that would otherwise be repeatedly inherited by downstream audit contexts.
- We propose BRA-Audit, which models MAS execution as a dynamic dependency graph, combines verification staleness and downstream reachability into a temporal–topological exposure measure, and greedily places a fixed number of audits by marginal reduction of cumulative unchecked exposure. Verified audit points further bound audit contexts and localize recovery.
- We evaluate BRA-Audit across structured coordination, complex reasoning, and open-ended collaboration. It achieves competitive recovery while using 17.2%–40.6% fewer tokens than strong auditing baselines, and same-budget comparisons confirm the benefit of cumulative-exposure-based placement over random auditing.

2 Related Work

Attacks and Failures in LLM-MAS. LLM-based multi-agent systems are vulnerable because agents exchange and reuse intermediate outputs, allowing local errors to escalate into system-level failures. Systematic analysis identifies failures in system design, inter-agent alignment, and

task verification, revealing collaboration-specific risks beyond isolated agents Cemri et al. (2025). Adversaries can exploit the same dependencies: malicious instructions can self-propagate across agents Lee & Tiwari (2024), manipulated knowledge can spread through conversations and persist in shared retrieval memories Ju et al. (2024), and communication attacks can intercept or modify inter-agent messages without compromising individual agents He et al. (2025). More severe control-flow hijacking can invoke unsafe tools, exfiltrate sensitive data, or execute malicious code Triedman et al. (2025). Overall, communication, shared memory, and orchestration amplify benign errors and adversarial inputs, making early detection and containment essential for reliable LLM-MAS.

LLM Multi-Agent Safeguarding. Early guards use prompt-based self-reflection, but the reviewer may share the generator's failure modes Shinn et al. (2023); Renze & Guven (2024). G-Safeguard applies a graph neural network to communication graphs to detect anomalous agents and prune risky links, yet relies on labeled attacks and may generalize poorly to unseen threats Wang et al. (2025b); Miao et al. (2026). More reliable approaches introduce LLM auditors. GuardAgent converts safety requirements into executable checks, TrinityGuard coordinates runtime monitors, PeerGuard uses mutual reasoning, and multi-agent debate performs iterative cross-examination Xiang et al. (2024); Wang et al. (2026); Fan & Li (2025); Du et al. (2024). These methods improve oversight, but their effectiveness and cost vary on complex tasks.

These safeguards methods either inspect trajectories too coarsely, which weakens detection and enlarges the rollback scope, or audit every interaction, which incurs substantial token and latency costs. We therefore propose BRA-Audit, a budgeted audit-point placement framework that preserves effective trajectory auditing while reducing redundant checks.

3 System Model

This section defines the LLM-MAS execution model and the threat model considered in this work.

3.1 MAS System

We consider a graph-structured LLM multi-agent system (LLM-MAS) with a set of agents

$$\mathcal{A} = \{a_1, \ldots, a_n\}. \tag{1}$$

Each agent has a predefined role and contributes to the task through its own interaction pattern, information sources, and tools. We represent runtime execution as a dynamic dependency graph:

$$G_t = (V_t, E_t). \tag{2}$$

Each node $v \in V_t$ represents a runtime object, such as an agent message, intermediate result, tool interaction, retrieved evidence, or aggregation result. It is associated with its producing agent $a(v)$, timestamp $t(v)$, and content $m(v)$. A directed edge $(u, v) \in E_t$ indicates that v depends on u through communication, evidence usage, tool execution, or aggregation. A path in G_t therefore captures the propagation of information and potential errors.

Auditing is modeled as audit-point placement on the execution graph. An audit point is a selected node $s \in V_t$ inspected by an auditor using the relevant context between the previous audit point and s. If the trajectory is safe, s becomes a trusted audit point in this round. Otherwise, the system may block, sanitize, isolate, or roll back the affected trajectory.

3.2 Threat Model

Unsafe behavior may arise from compromised agents, hallucinations, tool errors, poisoned knowledge bases, corrupted websites, or misleading retrieved documents. Despite their different causes, these failures introduce false, unsafe, or misleading information into the execution graph.

We use *malicious agents* as a unified abstraction for these cases. Let

$$\mathcal{M} \subseteq \mathcal{A}, \tag{3}$$

denote the set of malicious agents. For a node $v \in V_t$, its content may be unsafe if it is produced by a malicious agent:

$$a(v) \in \mathcal{M} \Longrightarrow \Pr(y(v) = 1) > 0. \tag{4}$$

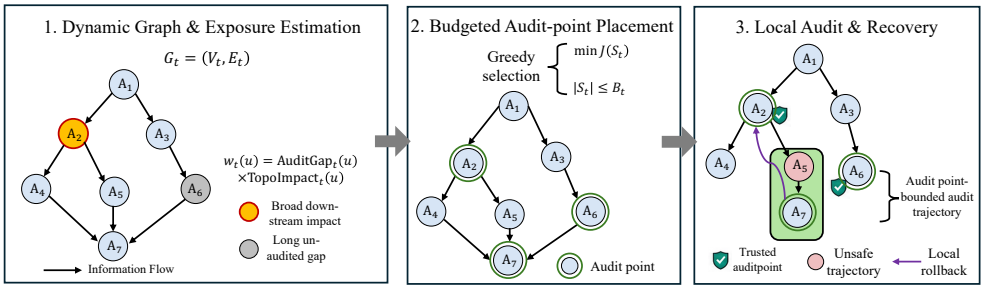


Figure 2: Overview of BRA-audit.

Here, $y(v) = 1$ denotes unsafe content. The term *malicious* does not require intentional behavior; it only indicates that an agent produces corrupted information that may mislead downstream agents.

A malicious agent may fabricate evidence, omit constraints, produce incorrect intermediate results, distort aggregation, or trigger unsafe tool usage. These outputs can propagate through dependency edges and affect later reasoning or the final result. We therefore study how runtime auditing detects and contains such propagated failures over the execution graph.

4 METHOD

This section we analyze the BRA framework: we formulate the budgeted runtime auditing as an optimization problem and develops the mathematical model and scheduling algorithm used by BRA-Audit (Fig. 2 shows overview of BRA-audit).

4.1 PROBLEM FORMULATION AND OPTIMIZATION OBJECTIVE

LLM-based auditors can inspect intermediate MAS outputs and identify hallucinated, inconsistent, or unsafe reasoning steps (Zheng et al., 2023; Gu et al., 2025). As discussed in the introduction, final-only auditing requires processing a long execution trajectory and may cause large rollback, whereas auditing every interaction requires frequent auditor calls. We therefore study how to allocate a limited audit budget during MAS execution.

A successful audit verifies the audit point-bounded dependency region reaching the inspected node and establishes the node as a trusted audit point in this round for subsequent execution. The size of this region determines both the context inspected by the auditor and the execution scope that may need to be rolled back. Audit-point placement is therefore important: leaving a node with broad downstream reachability unchecked can repeatedly carry its exposure into later audit contexts, while a long audit gap indicates stale trust evidence for the corresponding agent or region. Moreover, audit point decisions are coupled because each new audit point reshapes downstream audit regions and changes the value of subsequent audit points.

Given a candidate set $Q_t \subseteq V_t$ and an audit-call budget B_t, BRA-Audit selects before execution an audit point set

$$S_t \subseteq Q_t, \qquad |S_t| \le B_t. \tag{5}$$

Each selected audit point triggers one auditor call over its audit-point bounded dependency region. The scheduler therefore seeks placements that keep audit regions compact while prioritizing influential nodes and agents that have remained unaudited for many rounds.

We formulate audit point placement as minimizing cumulative unchecked exposure:

$$\min_{S_t \subseteq Q_t} J(S_t) \quad \text{s.t.} \quad |S_t| \le B_t, \tag{6}$$

where $J(S_t)$ measures the total cumulative exposure remaining under audit-point placement S_t. By preventing unchecked information from being repeatedly inherited by downstream contexts, this objective jointly reduces audit context size and delayed-detection exposure. Here, B_t controls the number of auditor calls, while total token cost is evaluated end to end. We define $J(S_t)$ and its components in the next subsection.

4.2 Analysis and Solution

We now analyze the cumulative-exposure objective and derive the audit-point scheduling rule used by BRA-Audit. Appendix C summarizes the main procedure of BRA-Audit. The key observation is that an unchecked node does not only affect its own audit context. Its exposure can continue to appear in downstream dependency contexts until the corresponding information flow reaches an audit point. Audit scheduling therefore considers how an audit point jointly reduces exposure in multiple downstream contexts.

Since runtime objects are ordered by their generation times and dependency edges follow causal information flow, repeated agent interactions are unrolled across rounds, and the runtime dependency graph G_t is represented as a directed acyclic graph.

For each node $u \in V_t$, let $a(u)$ denote its producing agent, and let $\tau_t(a)$ be the latest completed audit round of agent a available at scheduling time t. We define

$$\begin{aligned} \mathrm{AuditGap}(u) &= (t+1) - \tau_t(a(u)), \\ \mathrm{TopoImpact}_t(u) &= |\{u\} \cup \mathrm{Desc}_t(u)|, \\ w_{t+1|t}(u) &= \mathrm{AuditGap}(u)\,\mathrm{TopoImpact}_t(u), \end{aligned} \tag{7}$$

where $\mathrm{Desc}_t(u)$ denotes the strict descendants of u. We consider verification age because risk can accumulate while an agent's outputs remain unchecked, and downstream reach because errors at influential nodes can affect more subsequent computations and nodes with broad downstream reachability also tend to play more critical roles in the task. Including u itself ensures that even sink nodes retain nonzero local impact. Accordingly, the first term measures the verification age when the scheduled audit is executed, while the second measures the closed downstream region potentially affected by u. Their product serves as a temporal–topological exposure measure, prioritizing nodes with stale verification and broad downstream reach using runtime audit records and graph structure.

Let $\mathrm{Anc}_t(v)$ denote the ancestors of v. We write $u \rightsquigarrow_{S_t} v$ if there exists a directed path $p : u \rightsquigarrow v$ such that $(V(p) \setminus \{v\}) \cap S_t = \varnothing$. Given an audit point set S_t, the unchecked audit context reaching v is

$$\mathrm{Ctx}(v \mid S_t) = \{u \in \mathrm{Anc}_t(v) \cup \{v\} : u \rightsquigarrow_{S_t} v\}. \tag{8}$$

Equivalently, it is obtained by traversing backward from v and stopping each branch at the first selected audit point. This definition preserves unresolved branches in a DAG: if any path from u to v bypasses the selected audit points, u remains in the audit context. Thus, an audit point shortens only the downstream contexts whose information flow passes through it.

All audit points are selected before execution and triggered in causal order. Once an audit point passes the audit, its output becomes trusted input for downstream execution; an unsafe region is blocked and recovered before execution continues.

The cumulative exposure of v and the total exposure are

$$\begin{aligned} C(v \mid S_t) &= \sum_{u \in \mathrm{Ctx}(v|S_t)} w_t(u), \\ J(S_t) &= \sum_{v \in V_t} C(v \mid S_t). \end{aligned} \tag{9}$$

A high-exposure node can repeatedly contribute to downstream contexts until bounded by an audit point, increasing the audit burden and the cost of delayed detection. By summing over all nodes, $J(S_t)$ captures both the size of local audit contexts and the repeated inheritance of unchecked exposure by downstream computation.

Directly minimizing $J(S_t)$ is combinatorial, and audit point decisions are coupled because each selected audit point changes downstream contexts and the value of later selections. BRA-Audit therefore greedily selects audit points by marginal gain.

For a candidate $v \in V_t \setminus S_t$, let $\mathrm{Cut}_v(z \mid S_t)$ denote the exposure removed from the context of z:

$$\begin{aligned} \mathrm{Cut}_v(z \mid S_t) = \mathrm{Ctx}(z \mid S_t) \\ \setminus \mathrm{Ctx}(z \mid S_t \cup \{v\}). \end{aligned} \tag{10}$$

Table 1: Performance and token cost on BBH and Multi-agent-bench-rasearch.

	BBH			MultiAgentBench-Research					
Method	**Acc**	**recovery**	**token**	**innovation**	**safety**	**feasibility**	**Score**	**recovery**	**token(K)**
clean	0.9200	–	5355	5.00	4.62	3.83	0.897	–	23.0
attack	0.3933	–	6778	2.28	1.88	1.50	0.377	–	29.1
PeerGuard	0.8867	0.4933	10136	4.96	4.80	3.86	0.908	0.531	47.3
Auditagent	0.8067	0.4133	7459	3.88	3.92	3.14	0.730	0.352	32.4
Guardagent	0.5200	0.1267	7318	3.20	2.74	2.88	0.588	0.211	30.8
Trinityguard	0.8600	0.4667	8324	4.32	4.16	3.64	0.808	0.431	37.9
Multi-Agent Debate	0.9333	0.54	13763	4.98	4.86	4.02	0.925	0.548	60.5
BRA	0.8600	0.4667	8175	5.00	4.74	3.78	0.902	0.525	38.5

The marginal gain of v is

$$\begin{aligned}\Delta(v \mid S_t) &= J(S_t) - J(S_t \cup \{v\}) \\ &= \sum_{z \in V_t} \sum_{u \in \mathrm{Cut}_v(z|S_t)} w_t(u).\end{aligned} \tag{11}$$

This quantity measures the exposure removed from all dependency contexts by adding v. It is large when v separates a high-exposure region from many downstream contexts. Since marginal gains are recomputed after each selection, the scheduler accounts for overlapping effects among audit points rather than scoring them independently. For a chain, the expression reduces to the accumulated prefix exposure multiplied by the number of affected downstream contexts, we also illustrated its details in the technical appendix E.

At each step, BRA-Audit selects

$$v^\star = \arg\max_{v \in V_t \setminus S_t} \Delta(v \mid S_t), \tag{12}$$

adds $v^\star$ to S_t, and updates the affected contexts until the audit budget is exhausted.

4.3 Online audit-point Placement and Audit Execution

BRA-Audit operates in an online manner. At the end of each MAS interaction round t, the system updates the dependency graph $G_t = (V_t, E_t)$. Before the next round starts, the scheduler uses this graph to place a limited number of audit points. Let $Q_{t+1} \subseteq V_t$ denote the candidate nodes for the next audit round. Given an audit ratio $\rho \in (0, 1]$, which corresponds to auditing $k\%$ of the candidates, the audit budget and audit point set are defined as

$$B_{t+1} = \lfloor \rho |Q_{t+1}| \rfloor, \qquad S_{t+1} \approx \arg\min_{S \subseteq Q_{t+1},\, |S| \le B_{t+1}} J(S). \tag{13}$$

Here, $J(S)$ is the cumulative unaudited exposure objective defined above. In practice, BRA-Audit selects audit points that provide the largest reduction of $J(S)$ until the budget is used. In the final round, the agent producing the final result is mandatorily selected and counted toward the budget.

After the audit-point set S_{t+1} is selected, each audit point is assigned to an auditor. For an audit point $s \in S_{t+1}$, the auditor inspects the audit-point bounded dependency region accumulated along all incoming branches:

$$\mathcal{T}(s \mid S_{t+1}) = \mathrm{Region}(s \mid S_{t+1}). \tag{14}$$

Here, $\mathrm{Region}(s \mid S_{t+1})$ is obtained by traversing backward from s and stopping each branch at the first upstream audit point in S_{t+1}. The audit point outputs are treated as trusted inputs, while the subsequent inputs, outputs, messages, and dependency steps form the local audit region.

If the auditor judges $\mathcal{T}(s \mid S_{t+1})$ as safe, s becomes a trusted audit point; otherwise, the system rolls back the work products affected by this region. By partitioning execution into audit-point bounded regions, BRA-Audit shortens audit contexts and localizes rollback. Because the auditor examines the entire region rather than only its endpoint, an unsafe agent can still be detected even when it is not selected as an audit point, provided that its inputs or outputs appear in the region of an downstream audit point.

Table 2: Performance and token cost on AgentsNet. Token consumption is reported in thousands (K).

	Consensus		Leader		Matching		Coloring		VertexCover		Overall	
Method	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**
Clean	0.9630	28.1	0.5926	29.9	0.6944	25.7	0.9967	31.1	0.8342	18.6	0.8162	26.7
Attack	0.0741	42.2	0.1481	43.1	0.4815	35.7	0.8324	47.5	0.6676	29.0	0.4407	39.5
PeerGuard	0.9630	69.3	0.5556	67.5	0.8056	68.1	0.9716	70.4	0.8852	53.6	0.8362	65.8
AuditAgent	0.8148	48.6	0.4444	50.2	0.6250	49.1	0.9013	53.9	0.7854	40.3	0.7142	48.4
GuardAgent	0.4074	44.7	0.3704	46.5	0.5370	39.3	0.8548	50.2	0.7014	33.4	0.5742	42.8
TrinityGuard	0.8889	58.0	0.4815	55.6	0.7385	51.8	0.8892	64.8	0.8156	44.9	0.7627	55.0
Multi-Agent Debate	1.0000	75.4	0.6296	74.5	0.8342	77.3	0.9967	81.6	0.8675	71.5	0.8656	76.1
BRA-Audit	0.9630	57.6	0.5185	57.0	0.7917	52.4	0.9967	61.7	0.8342	43.8	0.8208	54.5

Table 3: Backbone model ablation on AgentsNet. Token consumption is reported in thousands (K).

	Consensus		Leader		Matching		Coloring		VertexCover		Overall	
Model	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**
Qwen	0.9630	57.6	0.5185	57.0	0.7917	52.4	0.9967	61.7	0.8342	43.8	0.8208	54.5
DeepSeekV4-Flash	0.9630	51.0	0.5556	72.9	0.6435	64.2	0.9463	51.3	0.8250	44.6	0.7867	56.8
MiniMax-2.7	0.9259	55.8	0.5185	63.4	0.7592	51.3	0.9177	55.4	0.7935	35.6	0.7830	52.3
GLM-5.1	0.9259	62.9	0.5926	63.2	0.7314	56.8	0.9446	61.0	0.8250	53.1	0.8039	59.4
GPT-5	0.9630	78.5	0.6296	79.8	0.8333	81.7	0.9534	75.9	0.8956	65.1	0.8550	76.2
Gemini3-Flash	0.8889	48.7	0.5556	51.5	0.7963	50.1	0.9265	53.7	0.8342	40.2	0.8003	48.8

5 EXPERIMENT

We evaluate BRA-Audit against recent LLM-agent auditing and multi-agent defense methods, focusing on defense effectiveness, audit cost, and robustness. Our experiments are designed to answer three research questions:

RQ1: Defense-Cost Trade-off. Can BRA-Audit achieve effective auditing while keeping the audit overhead low?

RQ2: Robustness Across Settings. How stable is BRA-Audit under different graph topologies, node importance distributions, attack locations, and backbone auditor models?

RQ3: Budget and Component Analysis. How do audit budget, audit point placement strategy, and exposure-weight components affect the overall performance of BRA-Audit?

5.1 EXPERIMENTAL SETUP.

Dataset. We evaluate BRA-Audit on three complementary settings. First, we use AgentsNet (Grötschla et al., 2025), a multi-agent coordination benchmark with five graph-theoretic task and configurable communication topologies and node counts. This setting tests auditing under controlled multi-round agent communication. Second, we use BIG-Bench Hard (BBH) (Suzgun et al., 2022) to evaluate complex logical reasoning; we randomly sample 150 questions and convert each instance into a multi-agent reasoning workflow. Third, we use the Research scenario from MultiAgentBench (Zhu et al., 2025), an open-ended multi-agent task where agents collaborate to generate research ideas.

Settings. We implement all multi-agent systems with AutoGen (Wu et al., 2023), and instantiate agent networks with different sizes and communication topologies. In each setting, 20% of the agents is designated as malicious and is instructed to inject incorrect claims, misleading evidence, or corrupted intermediate results into the collaboration process to make the task fail **(detailed attack recovery protocol is provided in Appendix G)**. We mainly use Qwen3-235B-A22B-Instruct-2507 model. We further evaluate different backbone models, graph topologies, numbers of agents, and audit budgets ρ. The main metrics are task score and total E2E token consumption, which can measure defense effectiveness and cost.

Table 4: Agent-count and topology ablations. Each entry reports Score / Avg. Token (K).

(a) Number of Agents

#Agents	Clean	Attack	BRA-Audit
4	0.859 / 8.3	0.363 / 14.1	**0.837 / 18.9**
8	0.827 / 21.3	0.520 / 33.4	**0.847 / 41.9**
16	0.763 / 50.4	0.439 / 71.0	**0.778 / 102.8**
Overall	0.816 / 26.7	0.441 / 39.5	**0.821 / 54.5**

(b) Graph Topology

Topology	Clean	Attack	BRA-Audit
WS	0.803 / 26.2	0.451 / 38.5	**0.846 / 55.6**
BA	0.835 / 25.2	0.415 / 38.1	**0.813 / 47.8**
DT	0.811 / 28.7	0.456 / 42.0	**0.804 / 60.1**
Overall	0.816 / 26.7	0.441 / 39.5	**0.821 / 54.5**

Table 5: Audit-budget ablation with audit and rollback costs. All token costs are reported in thousands (K).

	BBH				Research			
ρ	Score	Audit	Rollback	Total	Score	Audit	Rollback	Total
0.2	0.760	0.74	2.84	6.90	0.757	2.24	13.14	33.2
0.4, Rand.	0.820	2.14	1.76	8.01	0.821	5.43	9.13	37.4
0.4**, BRA**	**0.860**	2.09	1.81	8.18	**0.902**	5.82	9.38	38.5
0.6	0.873	3.91	1.35	9.57	0.906	9.11	7.17	42.9
1.0	0.893	5.57	0.94	11.31	0.897	11.13	5.64	46.8

Table 6: False-positive results under different audit ratios.

	BBH		Research	
ρ	FP	FPR	FP	FPR
0.2	0	0.00%	0	0.00%
0.4	7	4.67%	6	6.00%
1.0	18	12.00%	16	16.00%

5.2 Main Results.

We compare BRA-Audit with five representative LLM-based methods for runtime auditing or reliability enhancement. PeerGuard performs peer-based inspection during agent interactions (Fan & Li, 2025). AuditAgent is a round-level auditing baseline adapted from AgentAuditor (Yang et al., 2026). GuardAgent converts scenario-specific guard requirements into executable checks (Xiang et al., 2024). TrinityGuard uses preassigned runtime monitors to inspect MAS execution (Wang et al., 2026). Multi-Agent Debate improves output reliability through iterative discussion among agents (Du et al., 2024). All methods use the same attack settings and backbone models.

The attack results confirm that LLM-MAS is vulnerable to a small number of corrupted agents. Even one or two malicious or hallucinating agents can propagate faulty outputs through communication and aggregation, substantially degrading task performance. On AgentsNet, the overall score falls from 0.8162 in the clean setting to 0.4407 under attack, a relative drop of approximately 46.0%. This supports auditing MAS collaboration is necessary.

On AgentsNet (Table 2), BRA-Audit restores the overall score to 0.8208 with 54.5K tokens. It outperforms AuditAgent, GuardAgent, and TrinityGuard, and matches the clean results on Consensus, Coloring, and VertexCover. Although PeerGuard and Multi-Agent Debate achieve slightly higher scores, BRA-Audit uses 17.2% and 28.4% fewer tokens, respectively. A similar defense–cost trade-off appears on BBH and MultiAgentBench-Research (Table 1). BRA-Audit improves BBH accuracy from 0.3933 to 0.8600, using 19.3% fewer tokens than PeerGuard and 40.6% fewer than Multi-Agent Debate. On Research, it reaches 0.902 with 38.5K tokens, close to both baselines while reducing token usage by 18.5% and 36.3%, respectively. Overall, BRA-Audit balances defense effectiveness and runtime cost across structured and open-ended tasks.

5.3 Ablation Study

Backbone Model Ablation. We evaluate BRA-Audit with six backbone models on AgentsNet benchmark to study its sensitivity to model capability(as shown in Table 3.) Overall scores range from 0.7830 to 0.8550, indicating stable behavior across all tested models. GPT-5 obtains the highest score of 0.8550, but also incurs the largest token cost at 76.2K. Qwen offers a better performance–cost balance, reaching 0.8208 with 54.5K tokens. Gemini-3 uses the fewest tokens (48.8K) while maintaining a score of 0.8003. GLM-5.1, DeepSeek-V4, and MiniMax-2.7 achieve 0.8039, 0.7867, and 0.7830, respectively. Performance differences are most visible on Leader and Matching. Overall, BRA-Audit remains effective across different backbones, while stronger models tend to improve task performance at higher token cost.

Audit-Budget Ablation. We vary the audit ratio ρ on BBH and MultiAgentBench-Research to study the defense–cost trade-off. On BBH, increasing ρ from 0.2 to 0.4 improves the score from 0.760 to 0.860, while total token consumption rises from 6.9 to 8.18. Larger budgets of 0.6 and 1.0 further increase the score to 0.873 and 0.893, but require 9.57 and 11.31K tokens. MultiAgentBench-Research shows a similar trend: $\rho = 0.4$ achieves 0.902 with 38.5K tokens, close to the best score of 0.906 at $\rho = 0.6$. As ρ increases, audit cost grows while rollback cost decreases, indicating that more frequent auditing detects failures earlier and reduces re-execution. Under the same budget $\rho = 0.4$, BRA-Audit outperforms random audit-point placement by 0.040 on BBH and 0.081 on Research with comparable token costs. Overall, $\rho = 0.4$ provides a favorable balance among audit effectiveness, audit cost, and rollback overhead.

Node-Count Ablation. We evaluate BRA-Audit on AgentsNet graphs with 4, 8, and 16 agents (Table 4a). It achieves scores of 0.8370, 0.8473, and 0.7781, improving over the attacked setting by 0.4737, 0.3274, and 0.3391, respectively. Token consumption rises from 18.9K to 41.9K and 102.8K as larger graphs introduce longer communication paths and audit contexts. BRA-Audit therefore remains effective as the system scales, although larger networks require substantially more auditing resources.

Topology Ablation. We evaluate Watts–Strogatz, Barabási–Albert, and Delaunay graphs, which represent clustered small-world, hub-dominated, and spatially local communication structures, respectively (Grötschla et al., 2025). As shown in Table 4b, BRA-Audit obtains scores of 0.8458, 0.8127, and 0.8040, compared with attacked scores of 0.4511, 0.4153, and 0.4558. The small-world graph achieves the best score, while the preferential-attachment graph uses the fewest tokens. Overall, BRA-Audit remains effective across diverse communication topologies.

False-Positive Analysis. We assess false positives on clean BBH and MultiAgentBench-Research at varying audit ratios (Table 6). No FP occur at $\rho = 0.2$. At $\rho = 0.4$, FP rates are $4.67\%/6.00\%$ for BBH/Research, rising to $12.00\%/16.00\%$ under full auditing. Thus, excessive auditing can introduce unnecessary errors on benign trajectories. Most executions contain neither attacks nor severe hallucinations, so auditing every interaction incurs substantial redundant cost. Alongside task performance, BRA-Audit at $\rho = 0.4$ empirically lies on a favorable Pareto frontier balancing defense effectiveness, FP risk, and token consumption.

Weight-Component Ablation. We further ablate the two components of the exposure weight. Removing $\mathrm{AuditGap}$ decreases the overall score from 0.821 to 0.758 and increases token consumption from 54.5K to 57.2K, with particularly clear degradation on Leader and Matching. Removing $\mathrm{TopoImpact}$ similarly reduces the overall score to 0.793 while increasing the token cost to 59.1K; the effect is especially evident on the WS topology. These results suggest that $\mathrm{AuditGap}$ and $\mathrm{TopoImpact}$ provide complementary temporal and structural signals, respectively capturing verification staleness and downstream influence. Combining both components therefore improves audit-point selection in terms of both guard performance and token efficiency. Detailed results are provided in Appendix B.

6 Conclusion

In this paper, we presented BRA-Audit, a cost-aware runtime defense for LLM-based MAS. It models MAS communication as a dependency graph and formulates auditing as cumulative-exposure audit-point placement. By combining audit gaps with topological impact, BRA-Audit selectively places a limited number of audit points in long-unaudited and influential regions, thereby shortening audit contexts, limiting unchecked error propagation, and localizing rollback and recovery. This design enables the auditing budget to focus on interactions with greater potential risk and downstream influence, rather than uniformly inspecting all agent communications. Experimental results across structured coordination, complex reasoning, and open-ended tasks demonstrate a favorable Pareto trade-off between task performance and auditing cost, while maintaining effective recovery under diverse attack settings. Overall, BRA-Audit provides a practical framework for balancing runtime safety and computational efficiency, and we believe it can contribute to the development of more trustworthy, scalable, and cost-aware multi-agent systems.

REPRODUCIBILITY STATEMENT

We describe the system and threat models in Section 3 and present the cumulative-exposure objective and audit-point scheduling procedure in Section 4. Appendix C provides the algorithm pseudocode, while Appendices D–F detail audit-point selection examples, the objective rationale, and computational complexity. Section 5 presents the experimental settings and evaluations across benchmarks, backbone models, graph topologies, agent counts, and audit budgets. Additional component ablations are provided in Appendix B. Appendices G–H describe the attack and recovery protocol, benchmark settings, and evaluation metrics.

AI USE STATEMENT

We used AI tools to assist with running experiments and to polish the language of the manuscript for clarity and readability. The authors take full responsibility for the experimental results, analyses, and the final content of the paper.

REFERENCES

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2503.13657.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024.

Falong Fan and Xi Li. Peerguard: Defending multi-agent systems against backdoor attacks through mutual reasoning. *arXiv preprint arXiv:2505.11642*, 2025.

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, 2024. URL https://arxiv.org/abs/2411.04468.

Florian Grötschla, Luis Müller, Jan Tönshoff, Mikhail Galkin, and Bryan Perozzi. Agentsnet: Coordination and collaborative reasoning in multi-agent llms, 2025. URL https://arxiv.org/abs/2507.08616.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge, 2025. URL https://arxiv.org/abs/2411.15594.

Pengfei He, Yuping Lin, Shen Dong, Han Xu, Yue Xing, and Hui Liu. Red-teaming LLM multi-agent systems via communication attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6726–6747, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.349.

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. In *The Twelfth International Conference on Learning Representations*, 2024.

Tianjie Ju, Yiting Wang, Xinbei Ma, Pengzhou Cheng, Haodong Zhao, Yulong Wang, Lifeng Liu, Jian Xie, Zhuosheng Zhang, and Gongshen Liu. Flooding spread of manipulated knowledge in LLM-based multi-agent communities. *arXiv preprint arXiv:2407.07791*, 2024.

Donghyun Lee and Mo Tiwari. Prompt infection: LLM-to-LLM prompt injection within multi-agent systems. *arXiv preprint arXiv:2410.07283*, 2024.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008, 2023.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Rui Miao, Yixin Liu, Yili Wang, Xu Shen, Yue Tan, Yiwei Dai, Shirui Pan, and Xin Wang. BlindGuard: Safeguarding LLM-based multi-agent systems under unknown attacks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39215–39234, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.1819.

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810.

Matthew Renze and Erhan Guven. Self-reflection in LLM agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*, 2025.

Harold Triedman, Rishi Jha, and Vitaly Shmatikov. Multi-agent systems execute arbitrary malicious code. *arXiv preprint arXiv:2503.12188*, 2025.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024. URL https://arxiv.org/abs/2406.04692.

Kai Wang, Biaojie Zeng, Zeming Wei, Chang Jin, Hefeng Zhou, Xiangtian Li, Chao Yang, Jingjing Qu, Xingcheng Xu, and Xia Hu. Trinityguard: A unified framework for safeguarding multi-agent systems. *arXiv preprint arXiv:2603.15408*, 2026.

Liwen Wang, Wenxuan Wang, Shuai Wang, Zongjie Li, Zhenlan Ji, Zongyi Lyu, Daoyuan Wu, and Shing-Chi Cheung. IP leakage attacks targeting LLM-based multi-agent systems. *arXiv preprint arXiv:2505.12442*, 2025a.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on LLM-based multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7261–7276, Vienna, Austria, 2025b. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.359.

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 conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Yufei Xia, Anjun Gao, Yueyang Quan, Zhuqing Liu, and Minghong Fang. Who broke the system? failure localization in LLM-based multi-agent systems. *arXiv preprint arXiv:2607.07989*, 2026. To appear in the Conference on Language Modeling.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning. *arXiv preprint arXiv:2406.09187*, 2024.

Yizhe Xie, Congcong Zhu, Xinyue Zhang, Tianqing Zhu, Dayong Ye, Minghao Wang, and Chi Liu. Who's the mole? modeling and detecting intention-hiding malicious agents in LLM-based multi-agent systems. *arXiv preprint arXiv:2507.04724*, 2025.

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. *Frontiers of Computer Science*, 2026. doi: 10.1007/s11704-026-50857-y.

Wei Yang, Shixuan Li, Heng Ping, Peiyu Zhang, Paul Bogdan, and Jesse Thomason. Auditing multi-agent llm reasoning trees outperforms majority vote and llm-as-judge. *arXiv preprint arXiv:2602.09341*, 2026.

Can Zheng, Yuhan Cao, Xiaoning Dong, and Tianxing He. Demonstrations of integrity attacks in multi-agent systems. *arXiv preprint arXiv:2506.04572*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents, 2025. URL https://arxiv.org/abs/2503.01935.

A LIMITATIONS

BRA-Audit achieves a favorable empirical trade-off between audit effectiveness and runtime cost, but its current implementation has two main limitations. First, audit point scheduling for the next round uses the dependency graph observed in the current round. This causal design is suitable for workflows with moderate short-term structural continuity, but rapidly changing communication patterns may reduce the relevance of the selected audit points. Second, BRA-Audit introduces graph analysis and marginal-gain computation before auditing. This overhead is small relative to LLM inference in our evaluated settings, but its scalability to substantially larger and denser MAS remains to be studied. Incremental graph updates, uncertainty-aware scheduling, and distributed audit point selection are promising directions.

B ABLATION OF EXPOSURE-WEIGHT COMPONENTS

We ablate the two factors in the exposure weight,

$$w_t(u) = \mathrm{AuditGap}_t(u) \cdot \mathrm{TopoImpact}_t(u), \tag{15}$$

to examine their individual roles in audit point scheduling.

Effect of AuditGap. As shown in Table 7 removing $\mathrm{AuditGap}$ lowers the overall score from 0.8208 to 0.7578 and increases token consumption from 54.5K to 57.2K. The largest performance drops occur on Leader and Matching, where the scores decrease from 0.5185 to 0.4070 and from 0.7917 to 0.6546, respectively. Coloring also drops from 0.9967 to 0.9392, while Consensus remains unchanged.

A plausible explanation is that, without the temporal gap term, the scheduler repeatedly favors nodes with high structural influence. Less influential nodes may therefore remain between audit points for many rounds. Their evidence is then embedded in longer audit segments. LLM attention over long contexts can be uneven, and information in intermediate positions may receive less effective scrutiny Liu et al. (2024). This can reduce audit reliability even when the corresponding nodes are eventually included in a downstream audit trajectory. By increasing the priority of agents that have not been audited recently, $\mathrm{AuditGap}$ prevents persistent temporal blind spots and provides more balanced audit coverage.

Effect of TopoImpact. Table 8 shows that r emoving $\mathrm{TopoImpact}$ reduces the overall score from 0.8208 to 0.7932 and raises token consumption from 54.5K to 59.1K. The largest degradation occurs on the Watts–Strogatz topology, where the score falls from 0.8458 to 0.7652, while the token cost increases from 55.6K to 64.5K. This result suggests that downstream reachability is particularly useful when short communication paths allow one output to affect many subsequent interactions.

The effect is smaller on Barabási–Albert graphs, where the score decreases from 0.8127 to 0.8014. On Delaunay graphs, removing $\mathrm{TopoImpact}$ slightly increases the score from 0.8040 to 0.8131, but still raises token consumption from 60.1K to 62.1K. Thus, the performance gain of topology awareness varies with graph structure, but it improves overall audit efficiency.

Discussion. The two components capture complementary scheduling signals. $\mathrm{AuditGap}$ improves temporal coverage and prevents nodes from remaining unaudited for extended periods, whereas $\mathrm{TopoImpact}$ prioritizes nodes whose outputs can propagate widely through the dependency graph. Combining them achieves the best overall balance between task recovery and token cost.

C PSEUDOCODE FOR BRA AUDIT-POINT SELECTION

The detailed pseudocode for BRA audit-point selection is provided in Algorithm 1.

Table 7: Ablation of the AuditGap component on AgentsNet. Token consumption is reported in thousands (K).

	Consensus		Leader		Matching		Coloring		VertexCover		Overall	
Setting	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**
Clean	0.963	28.1	0.593	29.9	0.694	25.7	0.997	31.1	0.834	18.6	0.816	26.7
Attack	0.074	42.2	0.148	43.1	0.482	35.7	0.832	47.5	0.668	29.0	0.441	39.5
BRA-Audit	0.963	57.6	0.519	57.0	0.792	52.4	0.997	61.7	0.834	43.8	0.821	54.5
w/o AuditGap	0.963	51.0	0.407	72.3	0.655	64.2	0.939	54.8	0.825	43.6	0.758	57.2

Table 8: Ablation of the TopoImpact component across AgentsNet graph topologies. Tokens are reported in thousands (K). WS, BA, and DT denote Watts–Strogatz, Barabási–Albert, and Delaunay triangulation graphs, respectively.

	Clean		Attack		BRA-Audit		w/o TopoImpact	
Topology	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**	**Score**	**Tok.**
WS	0.803	26.2	0.451	38.5	0.846	55.6	0.765	64.5
BA	0.835	25.2	0.415	38.1	0.813	47.8	0.801	50.6
DT	0.811	28.7	0.456	42.0	0.804	60.1	0.813	62.1
Overall	0.816	26.7	0.441	39.5	0.821	54.5	0.793	59.1

D EXAMPLE OF AUDIT POINTS SELECTION

This appendix gives a chain-structured example to illustrate how BRA-Audit computes cumulative exposure and selects audit points. Consider a reasoning trajectory

$$a \to b \to c \to d \to e. \tag{16}$$

Each node $u \in \{a, b, c, d, e\}$ has an exposure weight w_u. The weight can be computed from the audit gap and topological impact defined in the main text:

$$w_u = \mathrm{AuditGap}(u) \cdot \mathrm{TopoImpact}(u). \tag{17}$$

A audit point cuts off the accumulated exposure before it from continuing into downstream audit contexts.

Initial cumulative exposure. If no audit point is placed inside the trajectory, the audit context grows along the chain. The cumulative exposure is

$$\begin{aligned} J(\emptyset) = w_a + (w_a + w_b) + (w_a + w_b + w_c) \\ + (w_a + w_b + w_c + w_d) + (w_a + w_b + w_c + w_d + w_e). \end{aligned} \tag{18}$$

Equivalently, this can be written as

$$J(\emptyset) = 5w_a + 4w_b + 3w_c + 2w_d + w_e. \tag{19}$$

This form shows that an unchecked upstream node can repeatedly appear in downstream audit contexts.

First audit point selection. We first compute the marginal gain of placing one audit point. If audit point x is selected, the exposure accumulated up to x will no longer enter the audit contexts of nodes after x. Therefore, the gain is the accumulated prefix exposure multiplied by the number of downstream nodes affected by this cut:

$$\Delta(x \mid \emptyset) = |\mathrm{Down}(x)| \sum_{u \in \mathrm{Pre}(x)} w_u. \tag{20}$$

Algorithm 1 BRA: Exposure Audit-Point Scheduling

Require: Execution graph $G_t = (V_t, E_t)$, candidate set $Q_{t+1} \subseteq V_t$, audit ratio ρ, and last audit records $\mathrm{LastAudit}_t(\cdot)$
Ensure: Selected audit points S_{t+1} and updated audit records $\mathrm{LastAudit}_{t+1}(\cdot)$

$B_{t+1} \leftarrow \lfloor \rho |Q_{t+1}| \rfloor$
$S_{t+1} \leftarrow \emptyset$
for each node $u \in V_t$ **do**
  $\mathrm{AuditGap}_t(u) \leftarrow t - \mathrm{LastAudit}_t(a(u)) + 1$
  $\mathrm{TopoImpact}_t(u) \leftarrow 1 + |\mathrm{Desc}_t(u)|$
  $w_t(u) \leftarrow \mathrm{AuditGap}_t(u) \cdot \mathrm{TopoImpact}_t(u)$
end for
while $|S_{t+1}| < B_{t+1}$ and $Q_{t+1} \setminus S_{t+1} \neq \emptyset$ **do**
  for each $v \in Q_{t+1} \setminus S_{t+1}$ **do**
    $\Delta(v \mid S_{t+1}) \leftarrow J(S_{t+1}) - J(S_{t+1} \cup \{v\})$
  end for
  $v^\star \leftarrow \arg\max_{v \in Q_{t+1} \setminus S_{t+1}} \Delta(v \mid S_{t+1})$
  $S_{t+1} \leftarrow S_{t+1} \cup \{v^\star\}$
end while
deploy S_{t+1} before round-$(t+1)$ execution
for each audit point $s \in S_{t+1}$ in causal order **do**
  $\mathcal{T}(s) \leftarrow \mathrm{Region}(s \mid S_{t+1})$
  $r \leftarrow \mathrm{Auditor}(\mathcal{T}(s))$
  if $r = \mathrm{safe}$ **then**
    mark s as a trusted audit point for downstream execution in this round
  else
    block the affected downstream information flow
    rollback and recover the work products affected by $\mathcal{T}(s)$
    re-execute and re-audit $\mathcal{T}(s)$ before continuing
  end if
  update $\mathrm{LastAudit}_{t+1}(a(s))$
end for
return $S_{t+1}, \mathrm{LastAudit}_{t+1}(\cdot)$

For the chain above, the gains are

$$\begin{aligned} \Delta(a \mid \emptyset) &= 4w_a, \\ \Delta(b \mid \emptyset) &= 3(w_a + w_b), \\ \Delta(c \mid \emptyset) &= 2(w_a + w_b + w_c), \\ \Delta(d \mid \emptyset) &= w_a + w_b + w_c + w_d. \end{aligned} \tag{21}$$

When all nodes have equal weight, $w_a = w_b = w_c = w_d = w_e = 1$, the gains are

$$\Delta(a \mid \emptyset) = 4, \quad \Delta(b \mid \emptyset) = 6, \quad \Delta(c \mid \emptyset) = 6, \quad \Delta(d \mid \emptyset) = 4. \tag{22}$$

Thus, either b or c is selected. This shows that under uniform exposure, the objective favors a balanced split of the trajectory. Suppose c is selected as the first audit point:

$$S_1 = \{c\}. \tag{23}$$

Cumulative exposure after the first audit point. With $S_1 = \{c\}$, the trajectory is split into

$$a \to b \to c, \qquad d \to e. \tag{24}$$

The cumulative exposure becomes

$$\begin{aligned} J(S_1) = w_a + (w_a + w_b) + (w_a + w_b + w_c) \\ + w_d + (w_d + w_e). \end{aligned} \tag{25}$$

For uniform weights, this gives $J(S_1) = 9$.

Second audit point selection. We now select a second audit point by recomputing the marginal gains under S_1. Since c has already cut the trajectory, candidates on the left and right segments are evaluated within their current segments.

On the left segment $a \to b \to c$, the remaining candidate audit points are a and b. Their gains are

$$\begin{aligned} \Delta(a \mid S_1) &= 2w_a, \\ \Delta(b \mid S_1) &= w_a + w_b. \end{aligned} \tag{26}$$

On the right segment $d \to e$, the remaining candidate audit point is d, with gain

$$\Delta(d \mid S_1) = w_d. \tag{27}$$

Under uniform weights, these gains are

$$\Delta(a \mid S_1) = 2, \quad \Delta(b \mid S_1) = 2, \quad \Delta(d \mid S_1) = 1. \tag{28}$$

Thus, the second audit point can be placed at a or b, further balancing the left segment. If b is selected, the final audit point set is

$$S_2 = \{b, c\}. \tag{29}$$

Effect of non-uniform weights. Now consider a non-uniform case where the later part of the trajectory has larger exposure:

$$w_a = 1, \quad w_b = 1, \quad w_c = 1, \quad w_d = 5, \quad w_e = 5. \tag{30}$$

The first-step gains become

$$\begin{aligned} \Delta(a \mid \emptyset) &= 4, \\ \Delta(b \mid \emptyset) &= 6, \\ \Delta(c \mid \emptyset) &= 6, \\ \Delta(d \mid \emptyset) &= 8. \end{aligned} \tag{31}$$

Thus, d is selected first:

$$S_1 = \{d\}. \tag{32}$$

This happens because the high exposure at d would otherwise continue into the audit context of e.

After selecting d, the trajectory is split into

$$a \to b \to c \to d, \qquad e. \tag{33}$$

For the second audit point, the remaining gains on the left segment are

$$\begin{aligned} \Delta(a \mid S_1) &= 3w_a = 3, \\ \Delta(b \mid S_1) &= 2(w_a + w_b) = 4, \\ \Delta(c \mid S_1) &= w_a + w_b + w_c = 3. \end{aligned} \tag{34}$$

Thus, b is selected as the second audit point:

$$S_2 = \{b, d\}. \tag{35}$$

This example illustrates two properties. First, when exposure weights are uniform, the algorithm tends to place audit points so that long trajectories are split more evenly. Second, when exposure weights are non-uniform, audit points move toward high-exposure regions to cut off their persistent burden on downstream audit contexts.

E RATIONALE OF THE BRA

This appendix expands on the cumulative-exposure objective used by BRA-Audit. We show how the objective reflects the accumulation of unchecked exposure along audit contexts, and why it leads to balanced audit points under uniform exposure and denser audit points near high-exposure regions.

Why cumulative exposure. Consider an unchecked trajectory segment

$$v_1 \to v_2 \to \cdots \to v_\ell. \tag{36}$$

Each node v_i has an exposure weight w_i. If no audit point is placed inside this segment, the audit context of v_i contains all unchecked nodes from v_1 to v_i. The cumulative exposure of the segment is therefore

$$J(v_1{:}v_\ell) = \sum_{i=1}^{\ell} \sum_{j=1}^{i} w_j. \tag{37}$$

Equivalently,

$$J(v_1{:}v_\ell) = \ell w_1 + (\ell - 1)w_2 + \cdots + w_\ell. \tag{38}$$

This form shows that an earlier unchecked node affects not only itself but also many later audit contexts. Thus, if an early node has a large exposure weight, leaving it unchecked creates a persistent burden for downstream auditing.

Why audit points reduce exposure. A audit point cuts the trajectory into shorter audit contexts. If a audit point is placed at v_k, then the segment is divided into

$$v_1 \to \cdots \to v_k, \qquad v_{k+1} \to \cdots \to v_\ell. \tag{39}$$

The exposure accumulated in $v_1, \ldots, v_k$ no longer enters the audit contexts of $v_{k+1}, \ldots, v_\ell$. This is the key reason why audit point placement is different from independent node ranking. A audit point is useful not only because the selected node is inspected, but also because it prevents preceding unchecked exposure from being repeatedly carried into later contexts.

Uniform weights lead to balanced placement. When all nodes have the same exposure weight, $w_i = 1$, a segment of length ℓ has cost

$$J(v_1{:}v_\ell) = 1 + 2 + \cdots + \ell = \frac{\ell(\ell+1)}{2}. \tag{40}$$

This cost grows superlinearly with the segment length. Therefore, a very long unchecked segment is more costly than several shorter segments with similar total length. As a result, when exposures are uniform, minimizing cumulative exposure naturally encourages audit points to be placed in a more balanced way. This matches the desired behavior: without additional risk differences, auditing should avoid long unchecked trajectories.

Non-uniform weights shift audit points toward high-exposure regions. When exposure weights are non-uniform, a high-weight node contributes to the audit contexts of all later nodes until it is cut off by a audit point. For example, if w_k is large and no audit point is placed at or after v_k, then w_k will repeatedly appear in downstream audit contexts. The objective therefore favors placing audit points near high-exposure regions, so that their exposure does not continue to burden later audits. This behavior follows from the cumulative structure of the objective, rather than from a manually weighted risk score.

Marginal-gain scheduling. The marginal gain of adding a audit point v to the current audit point set S is

$$\Delta(v \mid S) = J(S) - J(S \cup \{v\}). \tag{41}$$

This quantity measures how much cumulative exposure is removed by adding v. For a chain segment, if v splits the segment into a left prefix and a right suffix, the reduction can be written as

$$\Delta(v \mid S) = |\mathrm{Right}(v)| \sum_{u \in \mathrm{Left}(v)} w_u. \tag{42}$$

The first term counts how many downstream audit contexts are affected. The second term measures how much exposure is cut off. Therefore, a audit point has high marginal gain when it separates a large accumulated prefix from many downstream contexts.

Practical implication. The above structure leads to a simple scheduling rule: iteratively select the audit point that gives the largest marginal reduction in cumulative exposure, update the affected audit contexts, and repeat until the budget is exhausted. This rule follows directly from the structure of the objective: each selected audit point is the one that removes the largest amount of cumulative exposure under the current audit point set. It also avoids the main weakness of independent top-k scoring: after a audit point is selected, the value of other audit points changes because the audit contexts have changed.

Overall, the cumulative-exposure objective provides a structured way to combine audit coverage, graph influence, and rollback pressure. It explains why audit points should be balanced when exposure is uniform, and why they should move toward high-exposure regions when audit gaps or topological impacts differ.

F BRA-AUDIT COMPLEXITY

Let $n_t = |V_t|$, $m_t = |E_t|$, and $b_t = B_t$ denote the number of nodes, edges, and selected audit points at round t, respectively. Updating the execution graph incrementally requires $O(|\Delta V_t| + |\Delta E_t|)$ time. Computing $\mathrm{AuditGap}$ for all nodes takes $O(n_t)$ time. An exact computation of $\mathrm{TopoImpact}$ through graph traversal from each node has a worst-case complexity of $O(n_t(n_t + m_t))$. This cost can be reduced in directed acyclic execution graphs by caching descendant sets in reverse topological order.

BRA-Audit performs at most b_t greedy selection steps. Under a conservative implementation that recomputes the marginal gain of every candidate by traversing the affected contexts, audit point placement requires

$$O(b_t n_t(n_t + m_t)) \tag{43}$$

time. The total worst-case scheduling complexity is therefore

$$O((b_t + 1)n_t(n_t + m_t)), \tag{44}$$

with $O(n_t+m_t)$ memory when reachability is computed on demand. Caching reachability increases memory to $O(n_t^2)$, but avoids repeated graph traversals. For sparse graphs, where $m_t = O(n_t)$, the time complexity becomes $O(b_t n_t^2)$.

This bound is conservative. In practice, MAS execution graphs are typically sparse, and only newly added nodes and the descendants affected by a selected audit point need to be updated. BRA-Audit also uses only deterministic CPU-side graph operations and does not invoke an LLM during scheduling. At the scales evaluated in this work, the resulting overhead is small compared with a single auditor call, whose autoregressive LLM inference operates at a second-level latency. Therefore, the runtime is dominated by LLM auditing rather than audit point scheduling. Approximate reachability, lazy marginal-gain updates, and distributed graph processing can further improve scalability for substantially larger MAS.

G ATTACK PROTOCOL AND AUDIT POINT-BOUNDED RECOVERY

Attack protocol. For each episode, we designate a fixed subset of agents as compromised and instruct them to inject task-inconsistent outputs through the standard communication channel. The injected content is tailored to the task setting while preserving the original agent roles and interaction topology. In the Research scenario, compromised retrieval agents return irrelevant webpages, incorrect search results, or misleading evidence summaries. In BBH, compromised agents provide incorrect intermediate claims or deductions that steer the reasoning chain toward an incorrect answer. In AgentsNet, compromised agents send Byzantine messages that conflict with their local information or the required coordination protocol. These messages are consumed by downstream agents in the same manner as ordinary agent outputs, allowing us to evaluate whether corrupted information propagates through the execution graph.

Audit decision and fault attribution. For each selected audit point, the auditor receives the audit point-bounded trajectory, including the relevant inputs, messages, intermediate outputs, and dependency relations. It determines whether the trajectory remains consistent with the task objective and

identifies the earliest task-inconsistent message when an unsafe verdict is issued. We attribute the fault to the producing agent of this message and mark the corresponding message, the audited trajectory segment, and their dependent work products as untrusted.

audit point-bounded recovery. Upon an unsafe verdict, BRA-Audit rolls back to the latest preceding trusted audit point in this round. The attributed agent is replaced by a freshly initialized clean replica with the same functional role, system prompt, backbone model, decoding configuration, and tool permissions, but without the attack instruction or compromised post-audit point state. The system retains the trusted context and progress preceding the rollback boundary, invalidates the affected segment and its downstream dependent artifacts, and re-executes the workflow between the trusted audit point and the current audit point in this round. The regenerated trajectory is audited again; the current audit point is recorded as trusted only after the new trajectory receives a safe verdict. This procedure preserves verified progress while confining re-execution to the audit point-bounded region affected by the detected fault.

H BENCHMARK DETAILS

We evaluate BRA-Audit on three complementary benchmarks: AgentsNet, BIG-Bench Hard (BBH), and the Research scenario of MultiAgentBench. They cover structured coordination, complex multi-step reasoning, and open-ended collaboration, respectively. This section provides detailed descriptions of their task settings and evaluation metrics.

H.1 AGENTSNET

AgentsNet is a graph-based benchmark designed to evaluate coordination, communication, and collaborative reasoning in LLM multi-agent systems (Grötschla et al., 2025). Each agent corresponds to a node in a communication graph and can exchange messages only with its immediate neighbors. Execution proceeds in synchronous message-passing rounds. Agents must aggregate local information and coordinate their final decisions without a centralized controller.

AgentsNet contains five tasks derived from classical distributed computing problems:

Consensus. All agents must output the same binary value. This task evaluates whether local communication can lead to a consistent global decision.

Leader Election. Exactly one agent must identify itself as the leader, while all other agents must reject the leadership role. The task measures symmetry breaking and global coordination.

Matching. Agents select neighboring agents to form mutually consistent pairs. The final pairs must constitute a maximal matching: no agent can belong to more than one pair, and no additional valid pair can be added.

Coloring. Each agent selects one of the available groups, which correspond to graph colors. Adjacent agents must select different groups. This task tests decentralized role assignment and local conflict resolution.

Vertex Cover. Agents decide whether to act as coordinators. The selected agents must form a minimal vertex cover, so every communication edge is incident to at least one coordinator and no selected coordinator is redundant.

The benchmark includes three graph families. Watts–Strogatz small-world graphs have short average paths and high clustering. Barabási–Albert preferential-attachment graphs contain high-degree hubs and heterogeneous degree distributions. Delaunay triangulations represent spatially local communication and have relatively low node degrees. We evaluate graphs with 4, 8, and 16 agents to study the effects of both network scale and topology. For each task, the score is the fraction of graph instances satisfying the corresponding task-specific correctness conditions. The overall AgentsNet score is the mean score across the five tasks.

H.2 BIG-BENCH HARD

BIG-Bench Hard (BBH) is a collection of 23 challenging tasks selected from BIG-Bench because previous language models failed to outperform the average human rater. The benchmark covers a broad range of reasoning abilities, including logical deduction, algorithmic reasoning, arithmetic reasoning, symbolic manipulation, causal reasoning, and language understanding. Many instances require several intermediate reasoning steps rather than direct pattern matching.

We use BBH to evaluate runtime defenses in complex reasoning settings. Unlike AgentsNet, where success depends mainly on decentralized graph coordination, BBH requires agents to exchange, verify, and aggregate intermediate reasoning results. Consequently, an incorrect step produced by one agent can influence the reasoning of other agents and the final answer. This property makes BBH suitable for measuring whether auditing can detect and contain reasoning errors before they propagate through a multi-agent workflow.

Performance is measured by task accuracy:

$$\mathrm{Acc} = \frac{\text{number of correctly solved instances}}{\text{total number of evaluated instances}}. \tag{45}$$

We additionally report total token consumption, including the tokens used by task agents, defense agents, and auditor calls. The recovery score measures the absolute performance improvement over the attacked setting:

$$\mathrm{Recovery} = \mathrm{Acc}_{\mathrm{defense}} - \mathrm{Acc}_{\mathrm{attack}}. \tag{46}$$

H.3 MULTIAGENTBENCH-RESEARCH

MultiAgentBench evaluates LLM-based agents in interactive collaborative and competitive environments. We use its Research scenario to study BRA-Audit on an open-ended task without a single deterministic answer. In this scenario, multiple agents collaborate to generate a novel research idea. Each agent is assigned a specialized research profile constructed from an author's previous publications and research themes.

The original Research dataset contains 100 curated machine learning and artificial intelligence papers, including 33 easy, 34 medium, and 33 hard instances. The introduction of each paper provides the initial research context. Agents can use research tools to retrieve related or recent papers, search by keyword, title, or arXiv identifier, and inspect publication and coauthor information. They exchange findings and progressively refine a shared proposal.

The final output follows a structured five-question format:

1. What is the research problem?
2. Why is the problem important?
3. Why is the problem difficult?
4. Why has the problem not been solved previously?
5. What are the proposed approach and expected results?

The final research idea is evaluated on three dimensions using a five-point scale:

- **Innovation**: whether the proposal introduces novel concepts or advances beyond existing work;
- **Safety**: whether it adequately considers potential ethical, legal, and societal risks;
- **Feasibility**: whether it can be implemented with current methods, data, and computational resources.

We normalize the three ratings into an overall score:

$$\mathrm{Score} = \frac{s_{\mathrm{innovation}} + s_{\mathrm{safety}} + s_{\mathrm{feasibility}}}{15}. \tag{47}$$

The recovery score is defined as

$$\text{Recovery} = \text{Score}_{\text{defense}} - \text{Score}_{\text{attack}}. \tag{48}$$

We also report total token consumption across task execution and auditing. Compared with AgentsNet and BBH, this scenario contains longer interactions, heterogeneous agent expertise, tool use, and subjective output evaluation. It therefore tests whether BRA-Audit can generalize from structured coordination and deterministic reasoning to complex open-ended collaboration.

H.4 Complementarity of the Benchmarks

The three benchmarks evaluate distinct aspects of LLM-MAS execution. AgentsNet tests decentralized coordination under controlled graph structures. BBH evaluates the propagation of errors during complex multi-step reasoning. MultiAgentBench-Research measures defense effectiveness in long, tool-augmented, and open-ended collaboration. Together, they provide a broad evaluation of BRA-Audit across different task structures, communication patterns, and evaluation criteria.