TRACE-MAS: DIAGNOSING AND REPAIRING COLLABORATION FAILURES IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems (MAS) rely on not only individual agent behaviors but also the interactions that connect agents, including message exchanges and task dependencies. Existing MAS safeguards mainly perform agent-level anomaly detection, which becomes insufficient when failures originate from agent interactions and detected anomalies do not necessarily reveal their true sources. To address this limitation, we propose **TRACE-MAS**, a finer-grained framework for diagnosing and repairing MAS failures. TRACE-MAS learns normal collaborative execution patterns from benign trajectories and detects deviations in agent state changes, message transitions, and task dependency evolutions. It further identifies responsible execution elements through source-aware attribution and selects minimal interventions to restore collaboration while preserving task-critical capabilities. Experiments on five MAS benchmarks demonstrate that TRACE-MAS significantly improves root-cause localization accuracy by **60.2**% and achieves consistent gains in task recovery over strong MAS safeguards. Our code is available on https://anonymous.4open.science/r/x7k2m9f3.

1 INTRODUCTION

LLM-based multi-agent systems (MAS) coordinate specialized agents through role assignment, communication, tool utilization, and task planning, enabling complex applications such as software development (Hong et al., 2024), embodied decision making (Song et al., 2023), and collaborative scientific reasoning (Tang et al., 2024). However, unlike single-agent systems, MAS performance depends not only on individual agent behaviors, but also on the correct execution of interactions among agents. These dependencies introduce new attack surfaces: adversaries may inject malicious information into agent prompts or memories (Chen et al., 2024), manipulate messages exchanged between agents (He et al., 2025b), or interfere with task assignment and execution dependencies (Zhou et al., 2026b; Shahroz et al., 2025).

Recent MAS safeguards have substantially improved the detection and mitigation of compromised agents. Existing approaches mainly identify suspicious agents through interaction structures, observable behaviors, or system-level signals (Wang et al., 2025; Miao et al., 2026). However, these methods are primarily designed around agent-level anomaly detection: they treat individual agents as the basic detection units and answer the question of *which agent appears abnormal*. This coarse detection granularity becomes insufficient when failures originate from the interactions among agents rather than the agents themselves. For example, attackers can modify exchanged messages or disrupt task dependencies, causing normal agents to exhibit abnormal behaviors even though the actual corruption occurs in the collaborative process. Therefore, a key challenge is to identify corrupted execution elements at a finer granularity, while determining how the corresponding collaboration can be effectively recovered.

To investigate this challenge, we conduct a diagnostic empirical study in Sec. 4, revealing three connected observations. ❶ **Agent-Level Blind Spot.** Existing defenses are effective when attacks directly modify agent states. However, MAS execution can also be disrupted by modifying exchanged messages or altering task dependencies without changing individual agent profiles. As shown in Fig. 1, these execution-level attacks cause substantial task failures, indicating that agent abnormality alone cannot fully characterize MAS corruption. ❷ **Anomaly-Source Mismatch.** A natural solution is to extend monitoring beyond individual agents. However, broader anomaly de-

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.

tection still cannot determine where the failure originates. Once corrupted information propagates through normal collaboration, downstream components may exhibit stronger abnormal signals than the original source. As shown in Fig. 2, the most abnormal component is not necessarily where collaboration first deviates, demonstrating that anomaly localization and source identification are different objectives. ❸ **Repair-Target Mismatch.** Furthermore, the mismatch between anomaly and source directly limits recovery effectiveness, since repairing the most abnormal component may only remove propagated symptoms, while isolating suspicious components can unnecessarily eliminate useful capabilities. As shown in Fig. 3, source-aware repair achieves better recovery with fewer interventions than symptom-based strategies. Together, these observations show that existing MAS safeguards are limited by their agent-level detection granularity, which motivates a finer-grained modeling of collaborative execution beyond individual agents. Specifically, TRACE-MAS considers whether an agent's internal execution state changes abnormally, whether the information exchanged between agents is inconsistent with expected interactions, and whether task assignments or execution dependencies deviate from normal collaboration patterns.

Motivated by this perspective, we propose **TRACE-MAS**, a finer-grained framework for detecting, locating, and repairing MAS failures. TRACE-MAS models MAS execution as a structured event process and learns normal execution patterns from benign trajectories. It detects deviations in agent states, message transitions, and task dependency evolutions, identifies responsible execution elements through source-aware attribution, and selects minimal interventions to restore collaboration. By integrating detection, diagnosis, and recovery within a unified execution model, TRACE-MAS extends MAS safeguarding from identifying abnormal agents to fine-grained understanding and repairing corrupted collaborative execution.

Our Contributions. ***(1) New Problem.*** We study execution-level failure diagnosis and recovery in LLM-based MAS, where abnormal behaviors may originate from agent states, communication messages, or coordination operations. We reveal that detected anomalies and actual failure sources may not coincide, limiting existing agent-centric safeguards. ***(2) New Framework.*** We propose TRACE-MAS, which learns normal collaborative execution patterns from benign trajectories and performs unified execution-level detection, source-aware attribution, and minimal recovery. ***(3) SOTA Performance.*** Experiments on **5 benchmarks** covering reasoning, retrieval, and tool-use MAS under **3 attack settings** demonstrate that TRACE-MAS outperforms existing MAS safeguards, achieving an average improvement of **15.5 points.**

2 Problem Formulation

We consider an LLM-based multi-agent system with agents $\mathcal{A} = \{A_1, \ldots, A_N\}$ collaboratively solving a task q through role specialization, message exchange, and task dependencies. Instead of viewing agents independently, we model MAS execution as a partially ordered event process. Each event is represented as

$$e_t = \big(i_t, r_t, q_t, x_t, h_t, m_t^{\text{in}}, m_t^{\text{out}}, a_t, s_t\big), \tag{1}$$

where i_t, r_t, and q_t denote the agent, role, and subtask; x_t and h_t represent execution context and agent state; m_t^{in} and m_t^{out} denote exchanged messages; and a_t, s_t describe availability and task progress. The execution process is organized as a structured graph

$$\mathcal{G} = \big(\mathcal{V}, \mathcal{E}^{\text{msg}} \cup \mathcal{E}^{\text{dep}} \cup \mathcal{E}^{\text{temp}}\big), \tag{2}$$

where the edges capture message exchange, task dependency, and temporal order, and $\mathrm{Pa}(t)$ denotes the direct predecessors of e_t in $\mathcal{G}$. Let $\mathbf{o}_t = (\Delta h_t, \Delta m_t, \Delta s_t)$ denote the execution transition of event e_t, corresponding to changes in agent states, message interactions, and task progress. In implementation, textual and structured fields in Eq. (1) are embedded into the contract space as in Appendix A.1 (Eq. (14)); the transition residual Δe_t used in Sec. 5.2 denotes the same object as $\mathbf{o}_t$. Under benign collaboration, these transitions follow:

$$p_0(\mathcal{G} \mid q) = \prod_{e_t \in \mathcal{V}} p_0\left(\mathbf{o}_t \mid r_t, q_t, x_t, \mathbf{o}_{\mathrm{Pa}(t)}\right), \tag{3}$$

where p_0 describes normal execution dynamics conditioned on roles, tasks, and preceding events. Attacks can intervene on different execution elements, including agent states, message transitions, and task dependencies. We denote the resulting corrupted execution as $\widetilde{\mathcal{G}}$. Given benign trajectories

$\mathcal{D}_0 = \{\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(M)}\}$ and an observed execution $\widetilde{\mathcal{G}}$, the objective is to: *(i)* detect deviations from normal collaboration; *(ii)* identify the responsible execution element rather than propagated effects; and *(iii)* restore subsequent execution with minimal intervention.

3 Related Work

3.1 Attacks on LLM-based MAS.

LLM-based MAS inherit single-model risks such as jailbreaks and prompt injection (Perez et al., 2022; Wei et al., 2023; Greshake et al., 2023), and add new ones through agent interaction. AgentPoison corrupts agent memory and retrieval (Chen et al., 2024), AiTM and Prompt Infection show that inter-agent messages can carry and spread malicious content (He et al., 2025b; Lee et al., 2025), and recent attacks target coordination structures and execution dynamics (Zhou et al., 2026b), whose damage depends on the collaboration topology (tse Huang et al., 2025). These works analyze specific attack mechanisms, leaving open how to diagnose failures across heterogeneous collaborative executions.

3.2 Safeguarding LLM-based MAS.

Existing defenses detect suspicious behaviors and mitigate compromised components. G-Safeguard and NetSafe exploit the interaction topology (Wang et al., 2025; Yu et al., 2025), BlindGuard handles unknown attacks (Miao et al., 2026), AgentMonitor predicts risk from agent-level signals (Chan et al., 2024), and single-agent methods self-correct through execution feedback (Shinn et al., 2023; Yao et al., 2023). Failure-attribution studies catalogue MAS failure modes (Cemri et al., 2026) and locate the failing agent or step from logs (Zhang et al., 2025; Zhou et al., 2026a), but report where a failure becomes visible rather than repairing where it originates. TRACE-MAS complements these efforts by learning benign execution contracts, tracing the initiating violation, and repairing the source that disrupts subsequent execution.

4 Empirical Study

Before introducing TRACE-MAS, we investigate why existing MAS safeguards fail to achieve reliable diagnosis and recovery. We conduct controlled attacks on agent states, message exchanges, and task dependencies while keeping tasks and system configurations unchanged (detailed attack protocols are provided in Appendix B.4). As shown in Figs. 1–3, these studies reveal three connected observations: failures may originate beyond individual agents, observed anomalies may not indicate their sources, and effective recovery requires source-aware intervention.

Observation 1: Agent-centric monitoring misses relational corruption. Existing safeguards mainly associate MAS failures with abnormal agents. However, message exchanges and task dependencies can be disrupted without directly modifying agents. As shown in Fig. 1, when attacks shift from agent states to agent interactions, representative safeguards experience a **22.7-point** performance drop under comparable task failures, indicating that agent abnormality alone cannot characterize MAS corruption.

Observation 2: Observed anomaly does not reveal the root cause. Extending monitoring beyond agents does not directly solve diagnosis. Once corruption propagates through normal collaboration, downstream events may exhibit stronger deviations than the original source. Fig. 2 shows that the maximum-anomaly event identifies the true source in only **32.1%** of cases, with the strongest deviation appearing **0.7 hops** away from the source on average. Therefore, anomaly magnitude indicates where failure becomes visible rather than where it originates.

Observation 3: Symptom repair cannot guarantee recovery. The mismatch between anomaly and source directly limits recovery. Repairing the most abnormal event may leave the actual cause unchanged, while isolating suspicious components may remove useful capabilities. As shown in Fig. 3, source-aware repair improves task recovery by **30.0 points** over anomaly-based repair while requiring **50.0%** fewer interventions.

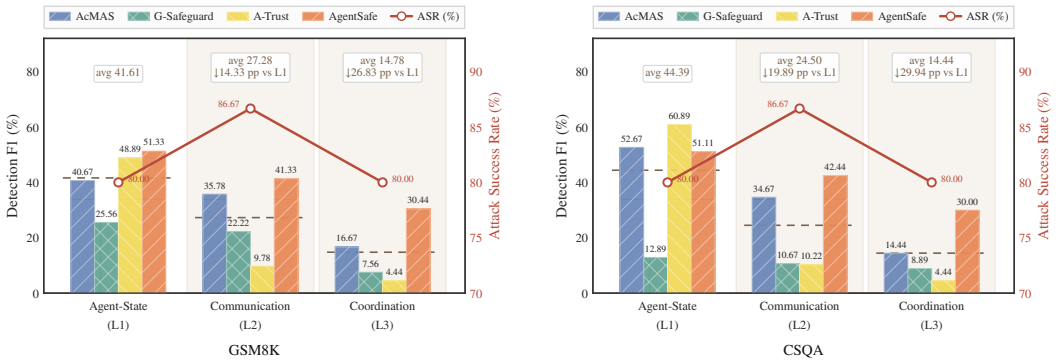


Figure 1: Agent-centric monitoring misses relational corruption. Detection F1 vs. L1–L3 on GSM8K and CSQA (matched ASR); F1 drops 14–30 points while ASR stays flat.

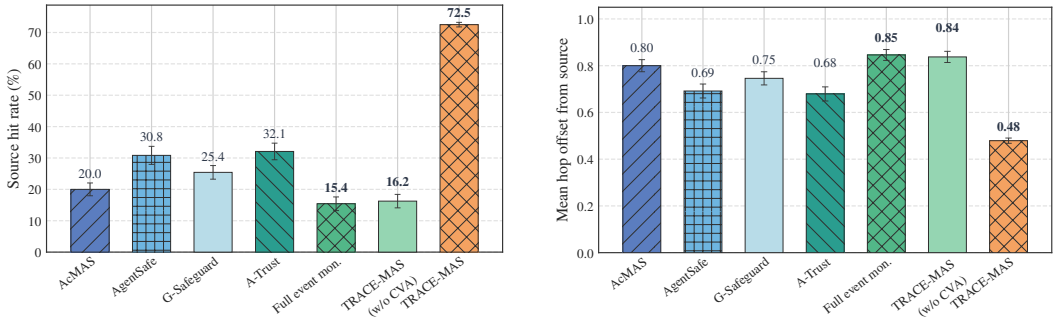


Figure 2: Observed anomaly does not reveal the root cause. Source hit rate and hop offset on HotPotQA; magnitude monitors $\leq$32.1%, TRACE-MAS 72.5%.

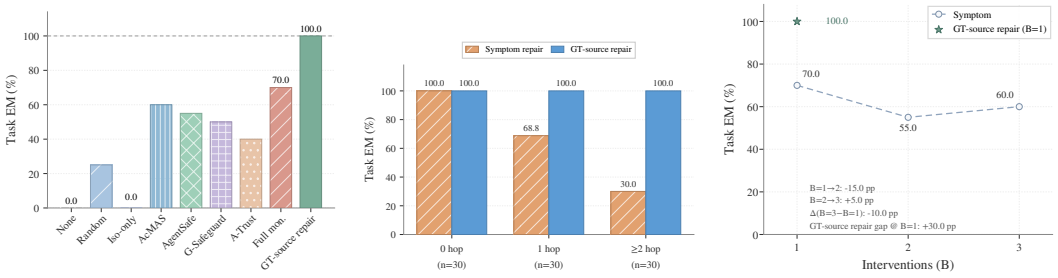


Figure 3: Symptom repair cannot guarantee recovery. Task EM, hop to source, and symptom budget B; source repair hits 100% EM, extra symptom steps do not help.

5 Methodology

5.1 Overview

As illustrated in Fig. 4, TRACE-MAS builds a single execution contract that characterizes how reliable collaboration should evolve, and reuses it for three purposes: learning the benign execution manifold conditioned on roles, tasks, and past interactions (Sec. 5.2); attributing observed deviations to the execution element whose restoration removes them (Sec. 5.3); and selecting the minimal intervention that returns future execution to that manifold (Sec. 5.4).

5.2 Execution Contract Learning

Motivation. Observation 1 shows that agent-level abnormality cannot characterize MAS failures, since correctness also depends on message interactions and task dependencies. TRACE-MAS therefore learns a conditional contract for how each event should evolve under its collaborative context.

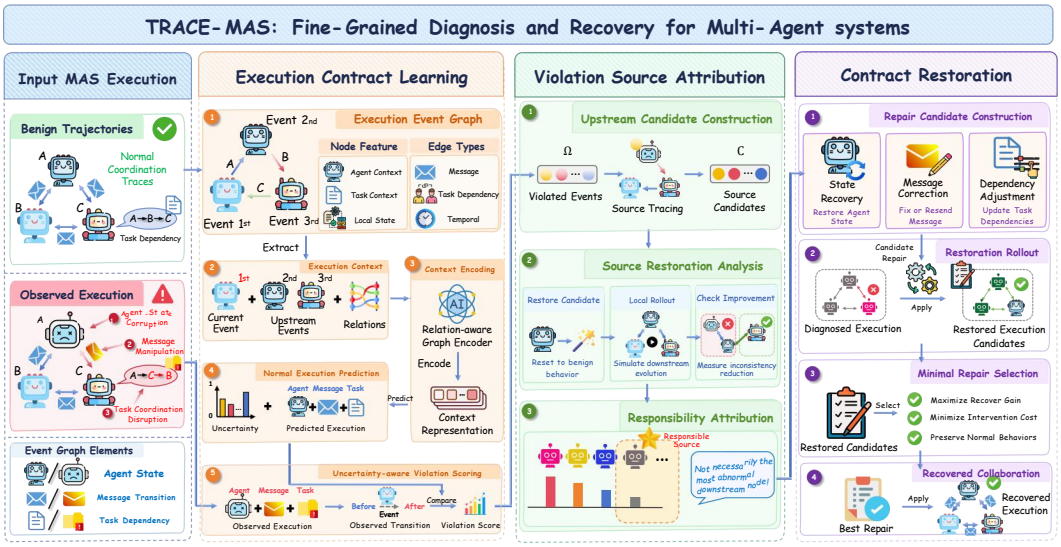


Figure 4: Overview of TRACE-MAS. Partial-order event graph and shared contracts for detection, source attribution, and repair.

Execution Context Encoding. TRACE-MAS encodes pre-execution information and available predecessors,

$$c_t = \mathrm{ExecFormer}\big(e_t^-, \{(e_j, \rho_{jt}) \mid e_j \in \mathrm{Pa}^-(t)\}\big), \tag{4}$$

where e_t^- contains only pre-execution fields, $\mathrm{Pa}^-(t)$ contains completed predecessors available before the step, and ρ_{jt} denotes the relation type. Current outputs and post-execution states are excluded from c_t and used only to construct the observed transition.

Uncertainty-aware Execution Contract. As roles and tasks legitimately vary, the contract is a conditional distribution fit on benign trajectories only:

$$p_\theta(\Delta e_t \mid c_t) = \mathcal{N}(\mu_t, \Sigma_t), \qquad \mathcal{L}_{\mathrm{con}} = -\sum_{G \in \mathcal{D}_0} \sum_{e_t \in G} \log p_\theta(\Delta e_t \mid c_t), \tag{5}$$

where $\Delta e_t = (\Delta h_t, \Delta m_t, \Delta s_t)$ collects the agent-state, message, and task-progress changes, with mean μ_t and adaptive uncertainty Σ_t. The violation score is

$$V(e_t) = (\widetilde{\Delta e}_t - \mu_t)^\top \Sigma_t^{-1} (\widetilde{\Delta e}_t - \mu_t), \tag{6}$$

where $\widetilde{\Delta e}_t$ is observed in the test trajectory $\tilde{G}$; its message and progress terms are weighted by (β, γ) (Appendix A.5), and $\tilde{G}$ is flagged when $\max_{e_t} V(e_t) > \tau$.

5.3 Violation Source Attribution

Motivation. Observation 2 shows that corruption propagates through benign interactions, so a downstream event can deviate more than the origin. TRACE-MAS therefore tests whether restoring a candidate recovers the subsequent execution.

Source Restoration Analysis. Violated events are $\Omega = \{e_t \mid V(e_t) > \tau\}$, with $\tau = \alpha \cdot \mathrm{Quantile}_\delta(\{V(e)\}_{e \in \mathcal{D}_{\mathrm{val}}})$ the α-scaled δ-quantile of benign validation scores (Appendix A.9). Each candidate $c \in \mathcal{C}$, an event of Ω or one of its H-hop ancestors, receives the contract expectation under its context c_c:

$$\Delta e_c^{\mathrm{res}} = \mathbb{E}_{p_\theta}[\Delta e_c \mid c_c] = \mu_c, \quad G_c^{\mathrm{res}} = \mathrm{Restore}_c(\tilde{G}, \Delta e_c^{\mathrm{res}}), \tag{7}$$

where $\mathrm{Restore}_c$ propagates the restored state downstream in contract space without LLM calls (Appendix A.6.2).

Responsibility Attribution. With $V_c^{\mathrm{res}}(e_j)$ the score recomputed on G_c^{res}, $\tilde{V}_c(e_j) = \omega V_c^{\mathrm{res}}(e_j) + (1-\omega)V(e_j)$, and $w_j = \psi^{\,d(c,e_j)}$ for hop distance d,

$$R(c) = \sum_{e_j \in \{c\} \cup \mathrm{Desc}(c)} w_j \big[V(e_j) - \tilde{V}_c(e_j)\big] - \lambda_{\mathrm{pen}} \mathcal{P}(c), \qquad c^\star = \arg\max_{c \in \mathcal{C}} R(c), \tag{8}$$

where $\mathrm{Desc}(c)$ are the strict descendants of c, $\psi, \omega \in (0, 1]$ are the hop decay and evidence gate, and $\mathcal{P}(c)$ penalizes dependency or ordering violations (Appendix A.6.3). $R(c)$ is thus the hop-discounted violation drop on the branch rooted at c, so the selected event explains the most propagated violation rather than deviating most.

5.4 Contract Restoration

Motivation. Observation 3 shows that repairing a visible anomaly may fail while the true source remains unchanged. TRACE-MAS therefore selects the intervention that returns future execution toward normality with minimal change.

Intervention-aware Future Rollout. Given $c^\star$, TRACE-MAS builds a repair space $\mathcal{U}(c^\star)$ (state recovery, message correction, dependency adjustment) and predicts the K-step future of each u from the repaired step t as $\hat{G}^u_{t:t+K} = \mathrm{Rollout}(\tilde{G}, u, p_\theta)$, with predicted events $\hat{e}^u_i$.

Minimal Contract Restoration. The committed intervention minimizes residual violation, task degradation, and edit cost,

$$u^\star = \arg\min_{u \in \mathcal{U}(c^\star)} \left[\sum_{i=t}^{t+K} V(\hat{e}^u_i) + \lambda_1 \mathcal{L}_{\mathrm{task}}(u) + \lambda_2 \kappa(u) \right], \tag{9}$$

where $\mathcal{L}_{\mathrm{task}}$ is the fraction of workflow-template steps the rollout leaves unsatisfied, $\kappa(u)$ the operator count, and λ_1, λ_2 trade-off weights (Appendix A.7.3); only the branches that $u^\star$ affects are regenerated.

6 Experiments

In this section, we conduct a comprehensive evaluation of TRACE-MAS to answer the following research questions. **Q1: Detection Performance.** Can TRACE-MAS accurately detect execution failures across different attack types? **Q2: Source Localization.** Can TRACE-MAS identify the true failure source rather than propagated anomalies? **Q3: Recovery Effectiveness.** Can TRACE-MAS restore collaboration with fewer interventions while preserving task performance? **Q4: Calibration and Robustness.** Can TRACE-MAS be calibrated from benign logs alone, remain stable under standard hyperparameters, and keep reliable detection under deployment shifts? **Q5: Runtime Overhead.** Is the non-LLM cost of monitoring, attribution, and repair scoring practical when layered on standard MAS rollouts?

6.1 Experimental Setup

We evaluate on five public LLM-based MAS benchmarks—GSM8K (Cobbe et al., 2021) and CSQA (Talmor et al., 2019) (multi-agent reasoning), HotPotQA (Yang et al., 2018) and PoisonRAG (Zou et al., 2025) (retrieval-based collaboration), and InjecAgent (Zhan et al., 2024) (tool use)—under controlled L1–L3 corruptions on shared role graphs and splits. We report trajectory-level detection, root-cause localization, and task-preserving recovery. Following prior attack–defense evaluations, we compare TRACE-MAS with interaction-graph monitors (BlindGuard (Miao et al., 2026), G-Safeguard (Wang et al., 2025)), trust- and agent-centric safeguards (A-Trust (He et al., 2025a), AgentSafe (Mao et al., 2025)), activation-centric MAS monitors(AcMAS(-R) (Xu et al., 2026)), and Oracle Anomaly Ranking. All methods share the same backbone LLM, MAS templates, attack injections, and benign-train / validation-calibration / attacked-test splits; baseline details, monitor variants, and calibration defaults are in Appendix B (Sec. B.5) and Appendix A.

6.2 Overall Performance (Q1)

To answer **Q1**, we compare TRACE-MAS with the baselines in Sec. 6.1 under the shared L1–L3 protocol. As shown in Table 1, TRACE-MAS achieves the highest trajectory F1 in every benchmark×level cell, improving over the strongest competing method by **15.5** points on average (per cell) and by up to **62.1** points on PoisonRAG at Level 3. Notably, TRACE-MAS remains above **52** F1 on GSM8K and CSQA under Level 2–3 while graph monitors and activation-centric methods often fall below **20** F1 when corruption targets messages or dependencies rather than agent states.

Table 1: Detection on five benchmarks under L1–L3 attacks (trajectory F1 and AUROC; mean±std). **Bold** best and underlined second-best per column; shared splits and calibration.

	Dataset	GSM8K		CSQA		HotPotQA		InjecAgent		PoisonRAG	
	Method	F1↑	AUROC↑	F1↑	AUROC↑	F1↑	AUROC↑	F1↑	AUROC↑	F1↑	AUROC↑
level 1	Oracle	$26.67_{\pm 0.40}$	49.67	$20.00_{\pm 0.41}$	52.62	$25.56_{\pm 0.92}$	50.69	$14.44_{\pm 0.87}$	49.27	$27.78_{\pm 0.92}$	51.96
	BlindGuard	$7.12_{\pm 2.61}$	65.00	$5.88_{\pm 0.14}$	51.47	$10.34_{\pm 2.81}$	54.05	$12.96_{\pm 0.26}$	54.96	$23.87_{\pm 0.58}$	54.48
	G-Safeguard	$26.00_{\pm 3.46}$	46.67	$26.67_{\pm 1.89}$	49.33	$25.08_{\pm 2.17}$	51.67	$25.56_{\pm 0.97}$	47.22	$22.22_{\pm 0.31}$	55.56
	AcMAS(-R)	$77.55_{\pm 3.04}$	91.84	$69.46_{\pm 0.97}$	90.88	$70.68_{\pm 0.78}$	52.01	$87.68_{\pm 0.79}$	91.79	$70.10_{\pm 0.74}$	88.30
	A-Trust	$68.63_{\pm 1.96}$	89.06	$64.59_{\pm 2.09}$	84.90	$43.81_{\pm 3.81}$	83.86	$58.20_{\pm 1.06}$	80.86	$46.82_{\pm 0.83}$	79.82
	AgentSafe	$68.26_{\pm 2.24}$	91.63	$67.77_{\pm 2.65}$	88.74	$67.97_{\pm 1.97}$	76.67	$69.94_{\pm 0.56}$	79.04	$69.14_{\pm 0.21}$	78.22
	Ours	**79.04**$_{\pm 3.88}$	**96.75**	**87.21**$_{\pm 0.36}$	**93.29**	**88.76**$_{\pm 0.18}$	**88.31**	**94.74**$_{\pm 0.35}$	**98.96**	**74.75**$_{\pm 0.16}$	**93.19**
level 2	Oracle	$35.56_{\pm 0.84}$	60.60	$22.22_{\pm 0.92}$	48.66	$26.67_{\pm 0.40}$	51.44	$24.44_{\pm 1.03}$	64.07	$22.22_{\pm 0.47}$	55.02
	BlindGuard	$10.29_{\pm 1.08}$	54.52	$8.73_{\pm 0.16}$	52.59	$16.52_{\pm 0.21}$	57.48	$11.10_{\pm 0.28}$	49.48	$9.33_{\pm 0.13}$	50.90
	G-Safeguard	$21.06_{\pm 0.37}$	50.44	$18.64_{\pm 0.34}$	55.19	$11.13_{\pm 0.24}$	47.81	$11.19_{\pm 0.21}$	46.29	$22.86_{\pm 0.31}$	55.19
	AcMAS(-R)	$23.33_{\pm 0.73}$	48.15	$16.29_{\pm 0.94}$	44.15	$16.89_{\pm 0.92}$	43.38	$15.85_{\pm 0.13}$	46.81	$26.25_{\pm 0.28}$	36.30
	A-Trust	$16.03_{\pm 0.64}$	60.42	$6.89_{\pm 0.47}$	39.58	$10.44_{\pm 0.36}$	33.85	$27.70_{\pm 0.56}$	50.26	$16.21_{\pm 0.49}$	48.18
	AgentSafe	$49.96_{\pm 0.16}$	38.67	$51.56_{\pm 0.73}$	45.11	$64.99_{\pm 0.32}$	37.70	$64.49_{\pm 0.74}$	42.63	$64.43_{\pm 0.29}$	43.19
	Ours	**52.96**$_{\pm 0.18}$	**63.84**	**52.84**$_{\pm 0.69}$	**63.79**	**67.43**$_{\pm 0.82}$	**61.28**	**67.68**$_{\pm 0.64}$	**70.73**	**66.55**$_{\pm 0.76}$	**68.28**
level 3	Oracle	$46.67_{\pm 0.42}$	63.90	$3.33_{\pm 0.49}$	14.69	$11.11_{\pm 0.96}$	15.94	$36.67_{\pm 0.96}$	55.80	$3.33_{\pm 0.28}$	10.37
	BlindGuard	$35.59_{\pm 0.36}$	59.26	$33.70_{\pm 4.58}$	58.72	$47.41_{\pm 0.29}$	68.37	$5.41_{\pm 0.03}$	54.74	$35.92_{\pm 0.43}$	62.67
	G-Safeguard	$19.86_{\pm 0.94}$	43.41	$15.84_{\pm 0.93}$	42.13	$16.96_{\pm 0.28}$	42.53	$66.67_{\pm 0.07}$	66.67	$22.37_{\pm 0.89}$	47.78
	AcMAS(-R)	$16.16_{\pm 0.88}$	44.59	$12.04_{\pm 1.60}$	37.92	$20.00_{\pm 0.27}$	44.42	$43.48_{\pm 0.18}$	73.33	$14.58_{\pm 0.92}$	42.07
	A-Trust	$1.24_{\pm 1.09}$	45.31	$4.44_{\pm 0.03}$	67.19	$14.42_{\pm 0.86}$	63.54	$14.07_{\pm 0.98}$	55.34	$3.03_{\pm 0.42}$	46.35
	AgentSafe	$30.44_{\pm 0.49}$	55.13	$30.08_{\pm 0.07}$	57.80	$30.00_{\pm 0.36}$	53.93	$26.84_{\pm 0.96}$	53.04	$2.67_{\pm 1.42}$	22.51
	Ours	**76.54**$_{\pm 0.38}$	**72.09**	**73.92**$_{\pm 0.47}$	**79.46**	**73.08**$_{\pm 0.64}$	**74.44**	**80.06**$_{\pm 0.02}$	**82.59**	**98.04**$_{\pm 0.71}$	**74.93**

Together, the table supports learning benign state, message, and completion contracts and flagging violations $V(e_t)$ on the event graph for reliable execution-level detection across corruption types.

6.3 Root-Cause Localization (Q2)

To answer **Q2**, we measure Hit@1 and hop distance to the labeled initiating event c^{gt} under the same injections as in Q1 (Table 2). TRACE-MAS ranks first on both metrics on every dataset, improving average Hit@1 by **28.0** points over the baseline with the highest average (AcMAS) and reaching **86.7**% Hit@1 with **0.18** hops on PoisonRAG. On HotPotQA it attains **71.1**% Hit@1 where the strongest activation-centric baseline remains at **23.3**%. The gains align with attributing failures via counterfactual violation drop $R(c)$ after contract `Restore`, instead of treating the largest downstream residual as the initiating event. Alert-pool diagnosis dominates the non-LLM cost at this stage; Q5 (Figure 6) shows that a small candidate cap C preserves most of this Hit@1 while substantially reducing diagnosis wall time.

Table 2: Root-cause localization: Hit@1 and mean hop distance to the labeled initiating event c^{gt}.

Dataset	GSM8K		CSQA		HotPotQA		InjecAgent		PoisonRAG	
Method	Hit@1↑	Distance↓	Hit@1↑	Distance↓	Hit@1↑	Distance↓	Hit@1↑	Distance↓	Hit@1↑	Distance↓
Oracle	$34.81_{\pm 0.64}$	$1.46_{\pm 0.01}$	$15.56_{\pm 1.11}$	$2.41_{\pm 0.07}$	$20.37_{\pm 0.68}$	$1.47_{\pm 0.03}$	$27.04_{\pm 0.64}$	$1.52_{\pm 0.02}$	$23.70_{\pm 1.28}$	$1.51_{\pm 0.04}$
BlindGuard	$2.31_{\pm 1.89}$	$2.33_{\pm 0.02}$	$1.33_{\pm 1.73}$	$2.48_{\pm 0.04}$	$1.60_{\pm 1.96}$	$2.60_{\pm 0.12}$	$2.22_{\pm 0.84}$	$3.78_{\pm 0.08}$	$4.45_{\pm 3.14}$	$1.17_{\pm 0.62}$
G-Safeguard	$31.00_{\pm 1.73}$	$0.38_{\pm 0.01}$	$46.67_{\pm 3.77}$	$0.41_{\pm 0.07}$	$45.33_{\pm 3.77}$	$0.76_{\pm 0.02}$	$75.55_{\pm 0.38}$	$0.40_{\pm 0.17}$	$31.11_{\pm 0.83}$	$0.43_{\pm 0.05}$
AcMAS	$48.00_{\pm 3.27}$	$0.59_{\pm 0.08}$	$54.67_{\pm 3.77}$	$0.44_{\pm 0.03}$	$23.33_{\pm 5.77}$	$1.01_{\pm 0.15}$	$55.55_{\pm 3.14}$	$1.04_{\pm 0.11}$	$51.11_{\pm 0.62}$	$0.65_{\pm 0.03}$
A-Trust	$21.67_{\pm 2.89}$	$1.05_{\pm 0.21}$	$15.55_{\pm 3.13}$	$1.32_{\pm 0.20}$	$15.00_{\pm 2.88}$	$0.72_{\pm 0.08}$	$4.38_{\pm 0.30}$	$2.75_{\pm 0.27}$	$4.37_{\pm 0.32}$	$1.95_{\pm 0.27}$
AgentSafe	$23.34_{\pm 0.57}$	$1.56_{\pm 0.11}$	$22.22_{\pm 0.62}$	$1.62_{\pm 0.03}$	$24.44_{\pm 0.63}$	$1.75_{\pm 0.03}$	$22.22_{\pm 3.14}$	$1.93_{\pm 0.05}$	$26.67_{\pm 0.54}$	$1.66_{\pm 0.09}$
Ours	**67.33**$_{\pm 0.94}$	**0.33**$_{\pm 0.01}$	**68.67**$_{\pm 0.94}$	**0.37**$_{\pm 0.02}$	**71.11**$_{\pm 4.16}$	**0.64**$_{\pm 0.04}$	**78.89**$_{\pm 0.57}$	**0.29**$_{\pm 0.03}$	**86.67**$_{\pm 0.27}$	**0.18**$_{\pm 0.01}$

6.4 Task-Preserving Recovery (Q3)

Repair results. To answer **Q3**, we evaluate post-alert recovery on attacked test trajectories under a **balanced** L1–L3 attack mix (pooled across levels, unlike the level-wise rows in Table 1). Table 3 reports ASR and task ACC after TRACE-MAS commits one repair at the attributed source; TCR and mean operator cost are in Appendix Table 9 (Appendix B.7.4). TRACE-MAS achieves the best ASR and ACC on every benchmark (e.g., PoisonRAG ASR **3.0**%, ACC **93.8**%), and Appendix Table 9 shows the strongest TCR–cost trade-off (PoisonRAG TCR **82.9**%, cost **0.54**), whereas retry-based recovery (AcMAS-R) and agent-level sanitization or trust-based exclusion (AgentSafe, A-Trust) lag

on utility or intervention cost under the same balanced-attack protocol. Predictive contract repair at $c^\star$ therefore restores collaboration with minimal intervention, whereas quarantining agents or patching downstream symptoms alone does not reach the same utility–cost balance.

Table 3: Post-defense ASR and task ACC on attacked trajectories.

Dataset	GSM8K		CSQA		HotPotQA		InjecAgent		PoisonRAG	
Method	**ASR↓**	**ACC↑**	**ASR↓**	**ACC↑**	**ASR↓**	**ACC↑**	**ASR↓**	**ACC↑**	**ASR↓**	**ACC↑**
Oracle	$65.19_{\pm 0.62}$	$47.78_{\pm 0.40}$	$84.44_{\pm 1.11}$	$31.85_{\pm 0.64}$	$79.63_{\pm 0.56}$	$44.81_{\pm 0.53}$	$71.48_{\pm 0.64}$	$62.22_{\pm 0.50}$	$76.30_{\pm 0.29}$	$23.70_{\pm 0.29}$
BlindGuard	$77.19_{\pm 0.16}$	$36.67_{\pm 0.47}$	$85.39_{\pm 0.67}$	$35.33_{\pm 0.47}$	$83.63_{\pm 1.12}$	$46.13_{\pm 0.75}$	$90.69_{\pm 0.36}$	$44.89_{\pm 1.66}$	$94.77_{\pm 1.12}$	$30.56_{\pm 0.79}$
G-Safeguard	$85.85_{\pm 0.38}$	$32.67_{\pm 0.43}$	$85.84_{\pm 1.30}$	$28.33_{\pm 1.70}$	$85.82_{\pm 0.99}$	$37.78_{\pm 1.26}$	$75.77_{\pm 0.30}$	$51.11_{\pm 1.26}$	$89.46_{\pm 2.83}$	$28.33_{\pm 2.72}$
AcMAS-R	$61.73_{\pm 0.18}$	$50.00_{\pm 0.12}$	$51.14_{\pm 0.33}$	$52.67_{\pm 1.25}$	$82.92_{\pm 1.06}$	$53.07_{\pm 1.36}$	$56.12_{\pm 1.25}$	$63.11_{\pm 1.66}$	$54.21_{\pm 1.13}$	$50.42_{\pm 0.72}$
A-Trust	$67.15_{\pm 1.81}$	$51.04_{\pm 1.47}$	$74.27_{\pm 0.51}$	$44.79_{\pm 1.48}$	$95.84_{\pm 0.42}$	$45.00_{\pm 0.41}$	$76.97_{\pm 0.46}$	$49.17_{\pm 0.31}$	$68.54_{\pm 0.24}$	$51.67_{\pm 1.18}$
AgentSafe	$6.05_{\pm 0.99}$	$83.89_{\pm 0.31}$	$38.63_{\pm 1.16}$	$83.34_{\pm 0.24}$	$23.81_{\pm 1.13}$	$73.33_{\pm 1.09}$	$4.55_{\pm 0.15}$	$70.22_{\pm 0.27}$	$4.67_{\pm 0.17}$	$75.11_{\pm 0.74}$
Ours	**3.33**$_{\pm 0.25}$	**96.00**$_{\pm 0.18}$	**29.33**$_{\pm 1.16}$	**84.62**$_{\pm 0.42}$	**18.42**$_{\pm 1.17}$	**96.67**$_{\pm 0.27}$	**2.98**$_{\pm 0.24}$	**82.89**$_{\pm 0.41}$	**3.02**$_{\pm 0.82}$	**93.76**$_{\pm 0.49}$

Ablation study. Building on the recovery results, we ablate the key designs of TRACE-MAS in Table 4, retraining each variant end to end on benign data with the same schedule as the full model. We replace the shared relation-aware contract with independent per-agent encoders, substitute absolute-embedding scoring for transition residuals, remove contract-based violation alerting, and replace rollout-based repair ranking with fixed operators. The resulting rows isolate how unified execution modeling, residual transition contracts, the alert–attribute loop, and predictive repair selection contribute to the detection, localization, and recovery trends reported in Q1–Q3.

Table 4: Ablation—one core module removed per row, each retrained end to end as Full and evaluated on balanced L1–L3 pooled splits.

Variant	Detect(F1)					Locate(Hit@1)					Repair(TCR)				
	GSM8K	**CSQA**	**HotPotQA**	**InjecAgent**	**PoisonRAG**	**GSM8K**	**CSQA**	**HotPotQA**	**InjecAgent**	**PoisonRAG**	**GSM8K**	**CSQA**	**HotPotQA**	**InjecAgent**	**PoisonRAG**
Full TRACE-MAS	**70.58**$_{\pm 0.37}$	**73.04**$_{\pm 0.44}$	**70.34**$_{\pm 0.96}$	**67.29**$_{\pm 1.05}$	**73.97**$_{\pm 0.29}$	**67.38**$_{\pm 0.54}$	**68.29**$_{\pm 0.22}$	**71.47**$_{\pm 0.90}$	**78.65**$_{\pm 0.58}$	**88.50**$_{\pm 0.66}$	**95.93**$_{\pm 0.67}$	**76.60**$_{\pm 0.37}$	**81.67**$_{\pm 0.79}$	**73.25**$_{\pm 0.27}$	**82.00**$_{\pm 0.36}$
Independent Contract	$60.77_{\pm 0.84}$	$62.09_{\pm 0.75}$	$60.16_{\pm 0.74}$	$59.32_{\pm 0.57}$	$62.71_{\pm 0.89}$	$51.17_{\pm 0.53}$	$53.37_{\pm 0.25}$	$56.93_{\pm 0.93}$	$44.68_{\pm 0.69}$	$53.88_{\pm 0.27}$	$55.14_{\pm 0.77}$	$55.79_{\pm 2.05}$	$53.23_{\pm 2.20}$	$39.16_{\pm 0.72}$	$55.47_{\pm 1.02}$
w/o Transition Contract	$63.69_{\pm 0.24}$	$65.95_{\pm 0.20}$	$63.50_{\pm 0.88}$	$60.82_{\pm 1.09}$	$66.72_{\pm 0.23}$	$54.93_{\pm 0.34}$	$58.10_{\pm 0.62}$	$61.49_{\pm 0.53}$	$48.20_{\pm 0.41}$	$58.50_{\pm 0.37}$	$60.18_{\pm 0.77}$	$61.24_{\pm 0.30}$	$58.26_{\pm 0.46}$	$42.37_{\pm 0.87}$	$60.94_{\pm 0.42}$
w/o Violation	$67.86_{\pm 0.31}$	$70.23_{\pm 0.34}$	$67.63_{\pm 0.93}$	$64.70_{\pm 1.01}$	$71.12_{\pm 0.22}$	$60.87_{\pm 0.75}$	$63.98_{\pm 0.73}$	$67.82_{\pm 0.45}$	$52.72_{\pm 0.96}$	$64.58_{\pm 0.55}$	$68.35_{\pm 1.01}$	$68.66_{\pm 0.39}$	$70.03_{\pm 0.40}$	$46.79_{\pm 0.96}$	$72.98_{\pm 0.54}$
w/o Predictive Repair	$69.22_{\pm 0.34}$	$71.63_{\pm 0.39}$	$68.99_{\pm 0.94}$	$65.99_{\pm 1.03}$	$72.55_{\pm 0.26}$	$62.62_{\pm 0.50}$	$66.40_{\pm 0.14}$	$69.52_{\pm 0.82}$	$75.12_{\pm 0.52}$	$86.61_{\pm 0.61}$	$67.81_{\pm 0.76}$	$69.03_{\pm 0.61}$	$65.63_{\pm 0.77}$	$47.66_{\pm 0.11}$	$68.75_{\pm 0.86}$

6.5 Calibration Convergence and Sensitivity (Q4)

To answer **Q4**, we study benign-only calibration, standard hyperparameter choices, and deployment stress factors.

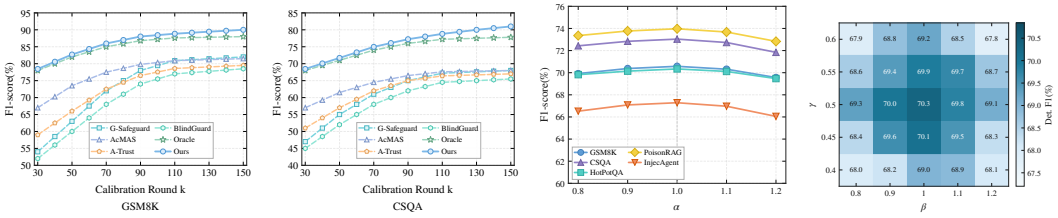
Benign-only calibration. We fit contracts and refresh the alert threshold τ from held-out benign trajectories only ($k \in [30, 150]$ calibration rounds; no attack labels for TRACE-MAS). Fig. 5(a–b) reports detection F1 on GSM8K and CSQA: TRACE-MAS gains F1 faster and saturates higher than interaction-graph, activation-centric, and trust-centric monitors.

Threshold scale α. We perform a hyperparameter sensitivity sweep on the alert scale α in $\tau = \alpha \cdot \mathrm{Quantile}_\delta(\cdot)$ over $\alpha \in \{0.8, 0.9, 1.0, 1.1, 1.2\}$, with (β, γ) fixed at their defaults and contracts calibrated as in the main protocol. Fig. 5(c) plots trajectory-level detection F1 against α, with **one line per benchmark** ; all curves peak at the default α=1 and change by only about **1–2** F1 points across the sweep, indicating mild sensitivity to threshold scaling on every workflow.

Contract weights (β, γ). We perturb the message- and completion-violation weights around $(\beta, \gamma) = (1, 0.5)$ on HotPotQA benign validation. Fig. 5(d) exhibits a wide high-F1 region near the default, indicating modest sensitivity to these weights in standard settings.

Deployment stress tests. We further stress-test detection under alert delay, agent scale, dynamic edges, and execution imbalance; full curves and baseline comparisons are in Figs. 7–11.

Together, the calibration and sensitivity panels in Fig. 5 support fitting TRACE-MAS from benign logs with a quantile threshold on $\max_t V(e_t)$, without attack labels during threshold selection.



(a) GSM8K, F1 (b) CSQA, F1 (c) F1 vs. α (d) F1 vs. β and γ

Figure 5: Benign calibration and hyperparameter sensitivity.

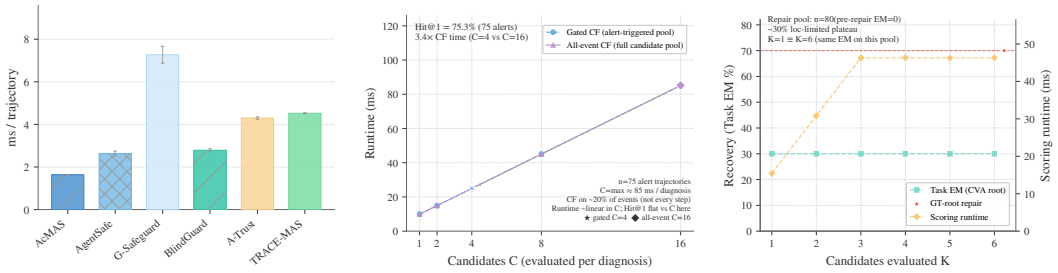
6.6 Runtime Overhead (Q5)

To answer **Q5**, we profile the non-LLM wall-clock cost of contract alerting, gated counterfactual diagnosis on the alert pool, and rollout-based repair scoring in Fig. 6 (bulk LLM generation is excluded because it dominates standard MAS rollouts and is invoked only for committed repairs).

Detection-stage overhead. Fig. 6(a) breaks down per-trajectory contract scoring and graph maintenance; this stage is lightweight relative to downstream diagnosis because it reuses cached event features from logging.

Alert-pool diagnosis. We vary the counterfactual candidate cap C on the alert-triggered pool. Fig. 6(b) shows that C=4 retains **75.3**% Hit@1 while cutting diagnosis time by about **3.4**× versus C=16, matching the gated pool design used after contract alerts in Q2.

Repair-pool scoring. We vary rollout depth K when ranking repair operators on the repair pool. Fig. 6(c) shows that TCR on the attack-successful repair pool of Q3 is unchanged from K=1 to K=6 while scoring cost grows with K, so we use K=1 (Appendix A.9).



(a) Detection (b) Diagnosis, C (c) Repair, K

Figure 6: Runtime overhead of TRACE-MAS.

7 Conclusion

In this work, we study collaboration failures in LLM-based multi-agent systems from an execution-centric perspective. We reveal that the attacked component, observed anomaly, and optimal repair target may not coincide during collaborative execution. To address this gap, we propose TRACE-MAS, which learns benign execution contracts and unifies violation detection, counterfactual root-cause attribution, and task-preserving recovery within a shared execution model. Experiments across diverse MAS benchmarks demonstrate that TRACE-MAS achieves more accurate failure diagnosis and effective recovery with minimal intervention. Future work will investigate adaptive execution contracts under open-ended environments where collaboration structures and agent behaviors continuously evolve.

AI USAGE DISCLOSURE

Generative AI tools were used during the preparation of this manuscript for limited purposes, including language refinement, literature retrieval and discovery, and research ideation support. Specifically, AI tools assisted with improving writing clarity, organizing related work, exploring potential research directions, and discussing possible methodological designs and experimental considerations.

REFERENCES

Anastasios N Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591, 2023.

Kailash Budhathoki, Lenon Minorics, Patrick Blöbaum, and Dominik Janzing. Causal structure-based root cause analysis of outliers. In *International conference on machine learning*, pp. 2357–2369. PMLR, 2022.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Chi-Min Chan, Jianxuan Yu, Weize Chen, Chunyang Jiang, Xinyu Liu, Xiaowei Chi, Weijie Shi, Zhiyuan Liu, Wei Xue, and Yike Guo. Agentmonitor: A plug-and-play framework for predictive and secure multi-agent systems, 2024. URL https://openreview.net/forum?id=gKM8wwsTOg.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. *Advances in Neural Information Processing Systems*, 37:130185–130213, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2024. URL https://openreview.net/forum?id=QAwaaLJNCk.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pp. 79–90, 2023.

Pengfei He, Zhenwei Dai, Xianfeng Tang, Yue Xing, Hui Liu, Jingying Zeng, Qiankun Peng, Shrivats Agrawal, Samarth Varshney, Suhang Wang, et al. To trust or not to trust: Attention-based trust management for llm multi-agent systems. *arXiv preprint arXiv:2506.02546*, 2025a.

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, 2025b.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Donghyun Lee, Mo Tiwari, and Brando Miranda. Prompt infection: Llm-to-llm prompt injection within multi-agent systems. In *European Symposium on Research in Computer Security*, pp. 511–520. Springer, 2025.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Junyuan Mao, Fanci Meng, Yifan Duan, Miao Yu, Xiaojun Jia, Junfeng Fang, Yuxuan Liang, Kun Wang, and Qingsong Wen. Agentsafe: Safeguarding large language model-based multi-agent systems via hierarchical data management. *arXiv preprint arXiv:2503.04392*, 2025.

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, 2026.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 3419–3448, 2022.

A Yang Qwen, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengpeng Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, pp. 2412–15115, 2025.

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

Rana Shahroz, Zhen Tan, Sukwon Yun, Charles Fleming, and Tianlong Chen. Agents under siege: Breaking pragmatic multi-agent llm systems with optimized prompt attacks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9661–9674, 2025.

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.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2998–3009, 2023.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621, 2024.

Jen tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael Lyu, and Maarten Sap. On the resilience of LLM-based multi-agent collaboration with faulty agents. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=bkiM54QftZ.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer, 2005.

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, 2025.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in neural information processing systems*, 36:80079–80110, 2023.

Haowen Xu, Xue Tan, Lei Ma, Zhihao Zhang, Chao Wang, Qingze Wang, Ping Chen, Jun Dai, and Xiaoyan Sun. When agents go rogue: Activation-based detection of malicious behaviors in multi-agent systems. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=BnduUW8izq.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. Netsafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2905–2938, 2025.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Jialong Zhou, Lichao Wang, and Xiao Yang. Guardian: Safeguarding llm multi-agent collaborations with temporal graph modeling. *Advances in Neural Information Processing Systems*, 38:7973–8001, 2026a.

Zhenhong Zhou, Zherui Li, Jie Zhang, Yuanhe Zhang, Kun Wang, Yang Liu, and Qing Guo. Corba: Contagious recursive blocking attacks on multi-agent systems based on large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6899–6908, 2026b.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 3827–3844, 2025.

A Methodology in Detail

This appendix expands the method of the main text. It keeps the same pipeline and the same symbols, and adds the parts that do not fit in the main paper: how execution logs become a typed event graph, how textual fields become transition features, how the execution contract is trained and calibrated, how candidate sources are constructed and scored, how repairs are ranked, and what each stage costs. Sec. A.8 summarizes the end-to-end procedure, and Sec. A.9 lists every default used in the main tables. Analytical support for the design choices is in Appendix D; shared experimental settings are in Appendix B.

Table 5 collects the symbols used throughout.

Table 5: Notation used in Appendix A and referenced by the main text.

Symbol	Meaning
$e_t = (i_t, r_t, q_t, x_t, h_t, m_t^{\text{in}}, m_t^{\text{out}}, a_t, s_t)$	event of Sec. 2: agent, role, subtask, context, state, messages, availability, progress
$G = (\mathcal{V}, E)$, $\tilde{G}$	event node set and edges; $\tilde{G}$ is the observed (attacked) graph
$\rho_{ij} \in \{\rho_{\text{msg}}, \rho_{\text{dep}}, \rho_{\text{temp}}\}$	message / task-dependency / temporal edge type
$\mathcal{H}_t$, c_t	ancestor neighborhood of e_t, encoded execution context
z_t^0, z_t, Δe_t	initial node features, contract-space embedding, transition residual
$p_\theta(\Delta e_t \mid c_t)$, $\mathcal{L}_{\text{con}}$	execution contract $\mathcal{N}(\mu_t, \Sigma_t)$ and its benign training loss
$\widetilde{\Delta e_t}$	transition observed in $\tilde{G}$ (scored against μ_t)
$V(e_t)$, τ, Ω	violation score, alert threshold, violated event set
φ_t	contract hidden state; the transition itself is Δe_t ($= o_t$ of Sec. 2)
$\mathcal{C}$, Anc_H, $\text{Desc}(c)$	candidate sources, H-hop ancestors, strict descendants of c (excluding c)
$V_c^{\text{res}}(\cdot)$, $R(c)$, $c^\star$	score after Restore_c, responsibility, attributed source
c^{gt}	labeled initiating event (evaluation only; never used by the method)
$\mathcal{U}(c^\star)$, $u^\star$, $\kappa(u)$	repair operators, selected repair, intervention cost
δ, α	benign quantile level and threshold scale in τ
$\omega, \psi, \lambda_{\text{pen}}$	restored-evidence gate, hop decay, validity penalty weight
β, γ	message- and completion-violation weights inside V (Fig. 5(d))
H, C, K	ancestor hop limit, scored candidates, rollout depth
$\hat{e}_i^u$, $\hat{\mathcal{J}}(u)$, $\mathcal{J}(u)$	rolled-out event, rollout estimate and realized value of the repair objective

A.1 Execution Graph Construction

A multi-agent execution consists of interleaved agent steps, messages, and task updates. TRACE-MAS represents each trajectory as a typed event graph that captures both local agent behavior and cross-agent coordination.

Each completed execution event is recorded as

$$e_t = \big(i_t, r_t, q_t, x_t, h_t, m_t^{\text{in}}, m_t^{\text{out}}, a_t, s_t\big), \tag{10}$$

following the event definition in Sec. 2. Here, i_t is the executing agent, r_t its workflow role, q_t its subtask, and x_t the task context available before execution. The fields h_t, m_t^{out}, and s_t record the agent state, emitted payload, and task progress after the step; m_t^{in} records the inputs consumed by the step, and a_t records agent availability. These fields are obtained from the benchmark execution logs without modifying the agents' prompts or decision rules.

To distinguish prediction inputs from observed outcomes, we additionally denote the information available immediately before the step by

$$e_t^- = \big(i_t, r_t, q_t, x_t^-, h_t^-, m_t^{\text{in}}, a_t^-, s_t^-\big), \tag{11}$$

where the superscript $-$ denotes a pre-execution snapshot. Pre-execution fields are read from logged snapshots or reconstructed from the execution prefix available before the step. They are never inferred from the current output or later events. In particular, e_t^- excludes the current outgoing message, newly generated scratchpad content, post-execution state, completion outcome, and any tool result produced during the step. A tool result may enter a subsequent event's context only after it has been returned.

Events become nodes of a partial-order graph

$$G = (\mathcal{V}, E), \tag{12}$$

whose edges carry relation types

$$\rho_{ij} \in \{\rho_{\text{msg}}, \rho_{\text{dep}}, \rho_{\text{temp}}\}. \tag{13}$$

The edge types are constructed by fixed rules: a ρ_{msg} edge connects the event that produces a message or tool result to the event that consumes it; a ρ_{dep} edge records an execution dependency, such as a verifier step requiring a solver output; and a ρ_{temp} edge connects consecutive events of the same agent. The workflow template specifies expected dependencies, while the event graph records the dependencies realized in the observed execution, including deviations introduced by corruption. Edges point from predecessors to successors, making G a directed acyclic graph.

This construction does not require lock-step rounds. Events are ordered by message consumption, execution dependencies, or an agent's own history. Delivery delays that preserve these relationships do not change the graph's dependency semantics, although they may change when information becomes available to the monitor. At each prediction step, only predecessor records already available to the monitor are used.

The three corruption levels affect different parts of this representation. L1 changes agent-state fields, L2 changes a transmitted message payload and the corresponding input received by its consumer, and L3 changes execution dependencies. The graph therefore provides a common representation for monitoring all three types of corruption.

For a completed event, the full node representation is

$$\begin{aligned} z_t^0 = [&f_i(i_t); f_r(r_t); f_q(q_t); f_a(a_t); f_x(x_t); f_h(h_t); \\ &f_m(m_t^{\text{in}}, m_t^{\text{out}}); f_s(s_t)]. \end{aligned} \tag{14}$$

This representation is used when the event serves as a completed predecessor of a later event. The current event instead uses a separate pre-execution representation,

$$\begin{aligned} z_t^- = [&f_i(i_t); f_r(r_t); f_q(q_t); f_a(a_t^-); f_x(x_t^-); f_h(h_t^-); \\ &f_m(m_t^{\text{in}}, \texttt{MASK}); f_s(s_t^-)], \end{aligned} \tag{15}$$

where MASK is a fixed placeholder independent of the current output. Structured fields use learnable embeddings, and textual fields are encoded as described in Sec. A.3. The same field layout is used for z_t^0 and z_t^-.

Received messages are legitimate inputs when predicting the recipient's response. Their inclusion does not establish the integrity of those messages: a payload must be assessed at its production or transmission event, without conditioning that assessment on the same payload as an observed outcome.

When multiple events are simultaneously eligible in the partial order, parsing uses deterministic tie-breaking by agent id and log line number. This convention selects a reproducible traversal without introducing additional dependency edges or making unavailable events visible.

A.2 Relation-Aware Execution Context Encoder

The execution contract predicts an event's transition from information available before that event executes. TRACE-MAS combines the current pre-execution representation with completed predecessor events through relation-aware attention.

Let $\mathcal{V}_t^{\text{avail}}$ denote the completed event records available to the monitor before step t. The context neighborhood is

$$\mathcal{H}_t = \{j \mid (e_j, e_t) \in E,\ e_j \in \mathcal{V}_t^{\text{avail}}\}. \tag{16}$$

Thus, an event cannot enter the context merely because it appears earlier in an offline serialization. It must be an available predecessor of the current event.

Each predecessor is transformed using parameters associated with its relation type:

$$k_{jt} = W_k^{\rho_{jt}} z_j^0, \qquad v_{jt} = W_v^{\rho_{jt}} z_j^0. \tag{17}$$

The complete representation z_j^0 is admissible here because event e_j has already completed. In contrast, the current query uses only the pre-execution representation:

$$\widetilde{q}_t = W_q z_t^-. \tag{18}$$

Attention weights are

$$\pi_{jt} = \frac{\exp\left(\widetilde{q}_t^\top k_{jt}/\sqrt{d}\right)}{\sum_{l\in\mathcal{H}_t} \exp\left(\widetilde{q}_t^\top k_{lt}/\sqrt{d}\right)}, \qquad j \in \mathcal{H}_t, \tag{19}$$

and the execution context is

$$c_t = \sigma\left(W_o \sum_{j\in\mathcal{H}_t} \pi_{jt} v_{jt} + W_s z_t^-\right). \tag{20}$$

For an empty neighborhood, the aggregated predecessor contribution is defined as zero. If two events are connected by multiple relation types, each typed edge contributes a separate key–value pair, with attention normalized over these pairs.

Both the query and the residual connection use z_t^-, so the current event's outgoing message, post-execution state, and completion outcome cannot enter its contract prediction. The residual term retains the agent's identity, role, task, and available local context. Post-execution fields are used only to construct the observed transition Δe_t and evaluate its violation score (Eq. 28). Holding the execution prefix and e_t^- fixed while changing the current output therefore leaves c_t and its predicted contract parameters unchanged.

Relation-specific projections allow the encoder to distinguish information received through messages, task dependencies, and an agent's own history. For example, a verifier consuming retrieved evidence may require a different expected transition from the same agent continuing its previous reasoning step. This provides a structured conditioning mechanism for the contract; its empirical contribution is evaluated through the relation-context ablation.

In implementation, one parent-event multi-head attention layer with four heads and hidden dimension $d = 256$ realizes this aggregation. Role and relation types use 64-dimensional learnable embeddings over nine role ids and $\{\rho_{\mathrm{msg}}, \rho_{\mathrm{dep}}, \rho_{\mathrm{temp}}\}$ (Table 6). With a fixed number of relation types and cached node projections, context encoding costs $O(|E|d + |\mathcal{V}|d^2)$ time. Node representations require $O(|\mathcal{V}|d)$ storage; storing the graph and attention weights additionally requires $O(|E|)$ memory. Events whose required predecessor records are available can be processed in batches.

A.3 Text-to-Transition Featureization

Logs are tokenized per event. Let ϕ_{text} be a frozen `all-MiniLM-L6-v2` Sentence-BERT (Reimers & Gurevych, 2019) encoder (384-dimensional pooled vectors) applied to the concatenated agent text of the event (thought, inbound and outbound message snippets, and task-state summary), followed by a learned linear map into the 256-dimensional contract space. Writing $z_t \in \mathbb{R}^{256}$ for the projected embedding of the active fields at e_t, the quantity modeled by the contract is the transition residual

$$\Delta e_t = z_t - z_{t-1}, \tag{21}$$

where z_{t-1} is the embedding of the immediately preceding state of the same execution thread. Here, z_t is the post-execution representation used to construct the transition target; it is not used in the current context c_t. Current-event context features are constructed exclusively from e_t^- as specified in Appendix A.2.

Modeling the residual rather than the absolute embedding is what makes benign-only training usable across benchmarks: absolute embeddings differ strongly between, say, a math scratchpad and a retrieved passage, while the *change* induced by one legitimate step is far more regular under a fixed role and relation type. The frozen encoder plays the same stabilizing role. Because it is never updated, the geometry of the feature space does not shift between the benign fitting split and the attacked test split, and no attack supervision can leak into the representation.

Three two-layer MLP heads (state, message, completion) predict the channel-wise mean μ_t, and the Gaussian head operates on the 256-dimensional residual block with a diagonal Σ_t. Following the main text, Δe_t denotes the transition variable of the contract, $\widetilde{\Delta e}_t$ its value observed in $\tilde{G}$, and μ_t the contract mean. At inference the implementation shrinks the observed residual toward the contract mean,

$$\bar{\Delta} e_t = \gamma_{\mathrm{res}}\, \widetilde{\Delta e}_t + \eta_{\mathrm{res}}\, \mu_t, \qquad (\gamma_{\mathrm{res}}, \eta_{\mathrm{res}}) = (0.5, 0.5), \tag{22}$$

before Mahalanobis scoring. Because $\gamma_{\mathrm{res}} + \eta_{\mathrm{res}} = 1$, this gives $\bar{\Delta} e_t - \mu_t = \gamma_{\mathrm{res}}(\widetilde{\Delta e}_t - \mu_t)$, so the scored value is exactly $\gamma_{\mathrm{res}}^2\, V(e_t)$ with V as in Eq. (28). Since τ is a quantile of the same scaled score, every alert, the set Ω, and every ranking of events are identical to those obtained from $V(e_t)$ itself; the shrinkage only fixes the numerical scale of the violation terms relative to λ_{pen}, λ_1, and λ_2, whose defaults (Sec. A.9) were set on this scale. All statements below and in Appendix D are therefore written for $V(e_t)$. The coefficient γ_{res} is distinct both from the completion-violation weight γ of Sec. A.5 and from the attribution hop decay ψ of Sec. A.6.3.

A.4 Probabilistic Execution Contract Learning

Given the context c_t, the contract states how a normal event evolves under that collaboration condition. The transition is decomposed by channel,

$$\Delta e_t = [\Delta h_t; \Delta m_t; \Delta s_t], \tag{23}$$

where Δh_t, Δm_t, and Δs_t are the changes in agent state, message interaction, and task progress. The decomposition is deliberate: it gives each corruption level a channel in which it is expressed, so a single scalar score can react to state edits, payload edits, and coordination edits without a separate detector per attack type.

A transition network predicts the expected evolution from a shared hidden state φ_t (kept distinct from the execution transition o_t of Sec. 2, which is the same object as Δe_t),

$$\varphi_t = \mathrm{MLP}_\theta(c_t), \qquad \mu_t = [g_h(\varphi_t); g_m(\varphi_t); g_s(\varphi_t)], \tag{24}$$

and, because legitimate executions are themselves variable, the contract also predicts a spread

$$\Sigma_t = \mathrm{Softplus}(W_\Sigma \varphi_t), \tag{25}$$

kept diagonal and floored away from zero for numerical stability. The contract is therefore

$$p_\theta(\Delta e_t \mid c_t) = \mathcal{N}(\mu_t, \Sigma_t). \tag{26}$$

Predicting Σ_t is what allows one threshold to serve heterogeneous roles. An agent that speaks rarely and an agent that speaks at every round have different benign transition spreads; with a shared fixed variance, the busier agent would accumulate large raw deviations and dominate the alerts. Conditioning the spread on the same context as the mean removes this volume effect, which is visible in the imbalance stress test of Appendix C.1.

Training uses benign trajectories only,

$$\mathcal{L}_{\mathrm{con}} = -\sum_{G \in \mathcal{D}_0} \sum_{e_t \in G} \log p_\theta(\Delta e_t \mid c_t), \tag{27}$$

where $\mathcal{D}_0$ is the normal execution set. In practice the objective is a weighted sum of the per-channel negative log-likelihoods for state, message, and completion, plus two auxiliary regularization terms, with weights λ_h=2, λ_m=1, λ_δ=2, λ_s=0.5, λ_p=0.5 (Table 6). Optimization uses AdamW (Loshchilov & Hutter, 2019) with learning rate 10^{-3}, batches of 8 trajectories, at most 100 epochs with early stopping (patience 20 on validation negative log-likelihood), 100 warmup steps, and gradient clipping at 1.0. The same configuration is used on all five benchmarks and is not re-tuned per dataset (Appendix B.7.2), so the detection numbers in Sec. 6 are not the product of per-benchmark contract tuning.

A.5 Violation Detection

At inference the realized transition is read from the observed trajectory, $\widetilde{\Delta e}_t = [\widetilde{\Delta h}_t; \widetilde{\Delta m}_t; \widetilde{\Delta s}_t]$, and compared against the contract channel by channel. Because Σ_t is diagonal, the Mahalanobis deviation of the main text splits exactly into a state, a message, and a progress term, $V_{\mathrm{ch}}(e_t) = (\widetilde{\Delta \mathrm{ch}}_t - \mu_t^{\mathrm{ch}})^\top (\Sigma_t^{\mathrm{ch}})^{-1} (\widetilde{\Delta \mathrm{ch}}_t - \mu_t^{\mathrm{ch}})$ for $\mathrm{ch} \in \{h, m, s\}$, and the violation score weights the last two:

$$V(e_t) = V_h(e_t) + \beta\, V_m(e_t) + \gamma\, V_s(e_t), \qquad (\beta, \gamma) = (1, 0.5). \tag{28}$$

At $\beta = \gamma = 1$ this is the unweighted form $(\widetilde{\Delta e}_t - \mu_t)^\top \Sigma_t^{-1} (\widetilde{\Delta e}_t - \mu_t)$; for general weights it is the same Mahalanobis form with the rescaled diagonal covariance $\tilde{\Sigma}_t = \mathrm{diag}(\Sigma_t^h, \Sigma_t^m/\beta, \Sigma_t^s/\gamma)$, since weighting a channel by β is identical to dividing its variance by β. (β, γ) are the message- and completion-violation weights swept in Fig. 5(d). This score is not a new objective introduced at test time: for the Gaussian contract of Eq. (26) it equals twice the negative log-likelihood of the observed transition under $\mathcal{N}(\mu_t, \tilde{\Sigma}_t)$ up to terms that depend only on the covariance (Appendix D.2). Training by Eq. (27) and scoring by Eq. (28) are thus the same Gaussian quantity evaluated on benign and on test trajectories, with the two channel weights acting as a fixed reweighting of its variance.

Events whose deviation exceeds the alert threshold form the violated set

$$\Omega = \{e_t \mid V(e_t) > \tau\}, \tag{29}$$

and a trajectory is flagged when $\max_t V(e_t) > \tau$, i.e. when $\Omega \neq \emptyset$. The threshold is fixed on benign validation trajectories,

$$\tau = \alpha \cdot \mathrm{Quantile}_{\delta}\big(\{V(e)\}_{e \in \mathcal{D}_{\mathrm{val}}}\big), \tag{30}$$

where δ selects the benign quantile and α rescales it into the operating threshold (main experiments: δ=0.99, α=1).

Calibration proceeds in three steps and uses no attack labels:

1. fit the contract on the benign training split (Sec. A.4);
2. score every event of the held-out benign validation split with Eq. (28) and refresh the benign quantile each calibration round ($k \in [30, 150]$ rounds, identical schedule for all methods, Appendix B.7.3);
3. set τ by Eq. (30) and keep it fixed for all attacked test trajectories.

Because τ is a benign quantile, the nominal false-alarm rate is controlled by δ by construction rather than tuned against attacks; Appendix D.2 states this property and its finite-sample form, and Fig. 5 reports how detection behaves as α moves around its default.

A.6 Violation Source Attribution

A.6.1 Candidate Source Construction

A violated event marks where inconsistency becomes visible, not necessarily where it began: once a corrupted payload enters the context of later events, those events are scored against contracts they can no longer satisfy. TRACE-MAS therefore searches upstream from Ω and builds a candidate set

$$\mathcal{C} = \bigcup_{e_i \in \Omega} \big(\{e_i\} \cup \mathrm{Anc}_H(e_i)\big), \tag{31}$$

where $\mathrm{Anc}_H(e_i)$ are the ancestors of e_i within H hops along $E_{\mathrm{msg}} \cup E_{\mathrm{dep}} \cup E_{\mathrm{temp}}$. The violated events themselves are included so that a source whose own deviation is already the largest (the 0-hop case of Fig. 3) stays eligible. Deployment uses a hop limit H=8 and scores the top C=4 candidates, ranked by their proximity-weighted violation mass; candidates are ordered deterministically so that ties do not depend on iteration order. Gating on Ω matters for cost: the number of counterfactual evaluations grows with C rather than with the trajectory length, which is the behavior profiled in Fig. 6.

A.6.2 Source Restoration Analysis

For each candidate $c \in \mathcal{C}$, the observed transition at c is replaced by the one the contract expects,

$$\Delta e_c^{\mathrm{res}} = \mathbb{E}_{p_\theta}[\Delta e_c \mid c_c] = \mu_c, \tag{32}$$

where c_c is the encoded context of candidate c, and the trajectory is re-evaluated on the resulting graph

$$G_c^{\mathrm{res}} = \mathrm{Restore}_c(\tilde{G}, \Delta e_c^{\mathrm{res}}). \tag{33}$$

Restore writes contract-consistent structured fields at c (message payload features, dependency flags, state summaries), recomputes z_c, and then refreshes the contexts of $\mathrm{Desc}(c)$ in embedding space with the same encoder used at detection time; the context c_c of the candidate itself is held at its pre-restoration value, so that μ_c is the target the restored transition is scored against. Two properties of this operation are used by the scoring rule below. It is *local*: only c and its descendants change, so scores on any branch that does not pass through c are unaffected. And it is *label-free*: the replacement value comes from the learned contract, never from the ground-truth source annotation c^{gt}, which is used only to evaluate Hit@1. Because c_c is held fixed, the restored transition equals its own contract mean and $V_c^{\mathrm{res}}(c) = 0$.

Counterfactual scoring makes no LLM call. Restoration and rescoring run entirely in the contract space, and message-correction operators invoke LLM re-generation only after $c^\star$ has been selected

and a repair is committed (Sec. A.7). This separation is why the diagnosis cost in Fig. 6 is measured without bulk generation, and why increasing C changes wall time but not the number of model calls in the underlying MAS.

A.6.3 Responsibility Estimation

A responsible source should explain a substantial part of the downstream violations once restored. Let $V_c^{\mathrm{res}}(e_j)$ be the score of e_j on G_c^{res}, and define the restored score and hop weights

$$\tilde{V}_c(e_j) = \omega\, V_c^{\mathrm{res}}(e_j) + (1-\omega)\, V(e_j), \qquad w_j = \psi^{\,d(c,e_j)}. \tag{34}$$

Responsibility and the attributed source are then

$$R(c) = \sum_{e_j \in \{c\} \cup \mathrm{Desc}(c)} w_j \big[V(e_j) - \tilde{V}_c(e_j)\big] - \lambda_{\mathrm{pen}} \mathcal{P}(c), \qquad c^\star = \arg\max_{c \in \mathcal{C}} R(c), \tag{35}$$

where $\mathcal{P}(c)$ penalizes candidates whose restoration would violate dependency or ordering constraints of the workflow template. Two conventions are worth stating explicitly. The sum ranges over c *together with* its descendants, and the candidate itself enters at hop distance $d(c,c) = 0$ and hence weight $\psi^0 = 1$; by $V_c^{\mathrm{res}}(c) = 0$ this own term equals $V(c)$. What separates an initiating event from a symptom on its branch is the descendant part of the sum: the source is credited with the drop its restoration causes at the symptom, whereas the symptom receives no credit for events upstream of it (Appendix D.4). And because $V(e_j) - \tilde{V}_c(e_j) = \omega\big[V(e_j) - V_c^{\mathrm{res}}(e_j)\big]$, the score is the hop-discounted counterfactual violation drop: the gate ω sets how much restored-branch evidence enters the ranking relative to the validity penalty, and at $\omega{=}1$ the drop is scored at full weight, so $c^\star$ is the candidate whose $\mathrm{Restore}_c$ removes the most violation mass. The hop decay ψ discounts far descendants, whose scores also depend on intervening events and are therefore weaker evidence about c. Values used in the main tables are in Sec. A.9. The scoring rule matches the responsibility objective in Sec. 5.3 of the main text.

A.7 Contract Restoration

A.7.1 Repair Space Construction

Attribution returns where to act, not what to do; several interventions are usually admissible at the same event. TRACE-MAS enumerates them as

$$\mathcal{U}(c^\star) = \{u_1, u_2, \ldots, u_{|\mathcal{U}(c^\star)|}\}, \tag{36}$$

with each candidate instantiating one intervention type: *state recovery* rewrites the agent state fields at $c^\star$ to contract-consistent values, *message correction* regenerates or replaces the payload carried by the outgoing ρ_{msg} edge, and *dependency adjustment* repairs the coordination structure (restoring a dropped dependency, reinstating a skipped role). The three types mirror the three corruption levels, so the repair space can express a fix at the level at which the failure was introduced instead of only suppressing its symptoms.

A.7.2 Future Execution Rollout

Each candidate is evaluated by how the collaboration would continue if it were applied,

$$\hat{G}^{u}_{t:t+K} = \mathrm{Rollout}(\tilde{G}, u, p_\theta), \tag{37}$$

where t indexes the attributed event $c^\star$. Rollout runs entirely in contract space and is fully specified by the four rules below.

Operator edit. Each operator edits one field of $c^\star$ and thereby one channel of its transition: state recovery edits the agent state h (channel h), message correction edits the outgoing payload m^{out} (channel m), and dependency adjustment edits the ρ_{dep} edges incident to $c^\star$ (channel s). We write $\mathcal{F}(u)$ for the edited field. In the rollout the edited channel of $c^\star$ is set to its contract mean, while the other two channels keep their logged values; for dependency adjustment the context of $c^\star$ is first recomputed on the edited edge set. The payload that message correction regenerates with the LLM exists only after commitment, so the rollout uses its contract mean in its place.

Window. The rolled-out events are the first K events of $\mathrm{Desc}(c^\star)$ in the deterministic topological order of Sec. A.1 (all of $\mathrm{Desc}(c^\star)$ if it has fewer than K events). We index them $t{+}1, \ldots, t{+}K$ in this order, so the sum in Eq. (9) of Sec. 5.4 runs over the edited event and the K window events.

Carry-over. Each relation type carries one field: ρ_{temp} carries the agent state (h), ρ_{msg} the payload (m), and ρ_{dep} the task dependency (s). A window event e_i is a *carry-over* event under u if it has a parent e_j such that either $e_j = c^\star$ and the field carried by ρ_{ji} is not $\mathcal{F}(u)$, or e_j is itself a carry-over window event. A carry-over event consumes a logged field that u leaves in place; every other window event is *regenerated*.

Recursion. For $i = t, t{+}1, \ldots, t{+}K$ in order, the context $\hat{c}_i^u$ is computed by the encoder of Sec. A.2 on the graph in which parents inside $\{c^\star\} \cup \{e_{t+1}, \ldots, e_{t+K}\}$ take their rolled-out features and all other parents keep their logged features. With $(\hat{\mu}_i^u, \hat{\Sigma}_i^u)$ the contract under $\hat{c}_i^u$, the rolled-out transition of a window event is

$$\widehat{\Delta e}_i^u = \begin{cases} \widetilde{\Delta e}_i & \text{if } e_i \text{ is a carry-over event,} \\ \hat{\mu}_i^u & \text{if } e_i \text{ is regenerated,} \end{cases} \qquad \hat{z}_i^u = \hat{z}_{i^-}^u + \widehat{\Delta e}_i^u, \tag{38}$$

where i^- is the preceding event of the same thread (rolled-out if it lies in the window, logged otherwise), and $\hat{z}_i^u$ is the feature later window events read. The rolled-out event $\hat{e}_i^u$ carries $\widehat{\Delta e}_i^u$, and its score $V(\hat{e}_i^u)$ is Eq. (28) with $(\hat{\mu}_i^u, \hat{\Sigma}_i^u)$ in place of (μ_i, Σ_i). Hence a regenerated event scores zero, a carry-over event contributes its logged transition rescored under the repaired context, and the edited event contributes the violation left in its two unedited channels. $\mathcal{L}_{\mathrm{viol}}(u)$ below is therefore the violation carried by the logged fields that u leaves in place within the window. Unlike $\mathrm{Restore}$, which only rescores the logged descendants of a candidate under refreshed contexts, Rollout also models which descendants would be regenerated once u is committed.

Commitment and real re-execution. Committing u applies the edit to the logged fields (message correction regenerates the payload with the trajectory LLM) and re-executes $\mathrm{Desc}(c^\star)$ with the same decoding (Table 8), leaving every event outside $\mathrm{Desc}(c^\star)$ untouched. Evaluating the three terms of the repair objective on the first K re-executed descendants, with the same contract, the same template steps, and the same κ, gives the realized objective $\mathcal{J}(u)$ that the rollout estimates. The two differ in exactly two places: the edited channel and the regenerated events are set to contract means rather than realized by the LLM, and carry-over events keep their logged transitions rather than being regenerated from partly repaired inputs. Appendix D.5 bounds the effect of both on the selected repair, and Appendix C.3 measures it by real re-execution of every candidate. Deployment uses $K{=}1$, because deeper rollouts did not change the top-1 repair in our runs, and Fig. 6 reports the cost of larger K.

A.7.3 Minimal Repair Selection

The selected repair minimizes remaining inconsistency, task degradation, and intervention cost jointly,

$$u^\star = \arg\min_{u \in \mathcal{U}(c^\star)} \big(\mathcal{L}_{\mathrm{viol}}(u) + \lambda_1 \mathcal{L}_{\mathrm{task}}(u) + \lambda_2 \kappa(u) \big), \tag{39}$$

with the three terms defined as

$$\mathcal{L}_{\mathrm{viol}}(u) = \sum_{i=t}^{t+K} V(\hat{e}_i^u), \tag{40}$$

$$\mathcal{L}_{\mathrm{task}}(u) = \frac{1}{|\mathcal{T}|} \sum_{s \in \mathcal{T}} \mathbb{1}\big[s \text{ unsatisfied in } \hat{G}_{t:t+K}^u\big], \tag{41}$$

$$\kappa(u) = \text{intervention cost (operator count / edit magnitude).} \tag{42}$$

Here $\mathcal{L}_{\mathrm{viol}}$ (the first term of Eq. (9) in Sec. 5.4, named here to keep it apart from the contract loss $\mathcal{L}_{\mathrm{con}}$) measures how much violation remains after the repair, with $\hat{e}_i^u$ the i-th rolled-out event under u, $\mathcal{L}_{\mathrm{task}}$ measures task degradation as the fraction of the workflow template's required steps $\mathcal{T}$ (roles to be invoked, dependencies to be satisfied, and their order) that the rolled-out graph leaves unsatisfied, read from the structured dependency and state fields that $\mathrm{Restore}$ and $\mathrm{Rollout}$ maintain, so it needs neither generated text nor gold labels, and $\kappa(u)$ charges for the edit itself; we write

κ rather than C to keep it distinct from the candidate cap C. Main runs use $\lambda_1 = \lambda_2 = 0.1$ unless noted. The third term is what keeps the reported operator cost low in Table 9: a repair that suppresses the attack by rewriting many events is penalized relative to one committed edit at the attributed source.

A.8 End-to-End Procedure and Cost

Putting the stages together, one attacked trajectory is processed as follows.

1. **Parse.** Build $\tilde{G}$ from the execution log with typed edges and deterministic tie-breaking (Sec. A.1).
2. **Score.** Encode contexts, evaluate the contract, and compute $V(e_t)$ for every event; raise a trajectory alert if $\max_t V(e_t) > \tau$ (Sec. A.5).
3. **Attribute.** Build $\mathcal{C}$ from Ω and $\mathrm{Anc}_H(\Omega)$, restore the top C candidates, and take $c^\star = \arg\max_c R(c)$ (Sec. A.6).
4. **Repair.** Enumerate $\mathcal{U}(c^\star)$, roll out K steps per candidate, select $u^\star$, and commit it with regeneration restricted to the edited branch (Sec. A.7).

The accounting of model evaluations follows directly: detection performs $|\mathcal{V}|$ contract evaluations, attribution performs at most $C \cdot |\mathrm{Desc}(c)|$ rescorings, repair performs at most $|\mathcal{U}(c^\star)| \cdot (K+1)$ rollout evaluations, and the only LLM calls are those of the committed repair. Since C, K, and $|\mathcal{U}(c^\star)|$ are small constants and detection reuses cached event features, the non-LLM overhead stays a small fraction of an MAS rollout, which is the behavior measured in Fig. 6.

A.9 Default Hyperparameters (Method)

Detection and repair defaults for the main tables are δ=0.99, α=1, ω=1, ψ=0.5, λ_1=λ_2=0.1, λ_{pen}=0.1, H=8, C=4, K=1; all five benchmarks share these values. The violation-channel weights of the detection score stay at (β, γ)=$(1, 0.5)$. Fig. 5 sweeps α and (β, γ) on held-out benign trajectories, and Fig. 6 varies C and K. Contract and encoder settings are listed in Table 6.

Table 6: ExecFormer and execution-contract training (all benchmarks).

Hyper-parameter	Value
Parent encoder	1 layer (parent-event MHA)
Contract heads	3× 2-layer MLP (state / message / completion)
Hidden dim. d	256
Attention heads	4
Role / relation emb.	64-d learnable (9 roles; $\rho \in \{\mathrm{msg}, \mathrm{dep}, \mathrm{temp}\}$)
Message encoder	`all-MiniLM-L6-v2` 384-d → Linear → 256
Transition space	$\Delta e_t \in \mathbb{R}^{256}$ (residual in contract space)
Optimizer / lr / batch	AdamW, 10^{-3}, 8 trajectories
Max epochs / early stop	100 (patience 20 on val NLL)
Warmup / grad clip	100 steps / 1.0
Multi-task loss weights	λ_h=2, λ_m=1, λ_δ=2, λ_s=0.5, λ_p=0.5
Inference residual mix	$(\gamma_{\mathrm{res}}, \eta_{\mathrm{res}}) = (0.5, 0.5)$

B Experiments in Detail

This appendix documents everything needed to reproduce Sec. 4–6: what is and is not compared (Sec. B.1), the five benchmarks and their task metrics (Sec. B.2), how trajectories are logged and split (Sec. B.3), how the L1–L3 corruptions are injected (Sec. B.4), how each baseline is instantiated on our testbed (Sec. B.5), how every reported quantity is defined (Sec. B.6), and the shared implementation, protocol, and randomness settings (Sec. B.7). Method-side defaults are fixed in Appendix A.9 and are not re-tuned per benchmark.

B.1 Evaluation Scope and Omitted Benchmarks

Controlled protocol (main results). All tables report defenses on the same five benchmarks with injected L1–L3 corruptions, shared MAS templates, and the metrics of Sec. 6, including task-preserving recovery. Every method sees the same trajectories, the same attack placement, the same benign calibration budget, and the same repair operators where applicable, so differences in the tables reflect the monitoring and attribution logic rather than differences in workflow, decoding, or attack strength. The protocol is therefore scoped to injected corruptions with a known initiating event, which is what makes the localization and recovery metrics of Sec. B.6 computable at all.

Why natural failure corpora (Who&When (Zhang et al., 2025), MAST-Data (Cemri et al., 2026)) are not used as benchmarks. Our three metric groups each require an input that these corpora do not release, so the exclusion follows from the metric definitions rather than from a judgement about the corpora. (i) *Contract inputs.* Every score in this paper is conditioned on the encoded context c_t, which is built from the typed edges ρ_{msg}, ρ_{dep}, and ρ_{temp} of Appendix A.1. These corpora release conversation turns or taxonomy-labeled steps; which event consumed which message, and which task depends on which, are not recorded, so $\mathcal{H}_t$ cannot be recovered from the released logs and Eqs. (26)–(28) are not defined on them without re-annotating every trace. (ii) *Localization labels.* Hit@1 and hop distance (Sec. B.6) are defined against a single labeled *initiating event* c^{gt} per trajectory. The released labels mark a failing agent, a failing step, or a failure category, which identifies where the failure is visible rather than where it originates; by Obs. 2 those two need not coincide, so the labels cannot be reinterpreted as c^{gt} without new annotation. (iii) *Recovery metrics.* ASR, ACC, TCR, and operator cost all require re-executing the workflow after a repair is committed. These corpora are static traces without an executable workflow behind them, so no operator from $\mathcal{U}(c^\star)$ can be applied and no post-repair task metric can be computed. (iv) *Calibration split.* Our thresholds are fit by Eq. (30) on a benign split with task ids disjoint from the test split, and cross-method comparison relies on that split being shared (Sec. B.7.2). These releases do not provide a matched benign split from the same workflow, so no method in Table 1 could be calibrated on them under the protocol used for the reported numbers. Because (i)–(iv) would each require new annotation or new instrumentation, any number we produced on these corpora would be measured under a different protocol from the rest of the paper; we therefore report none and treat the extension as future work.

Why attribution pipelines such as AutoFA (Zhang et al., 2025) and GUARDIAN (Zhou et al., 2026a) are not rows in Table 1. The same protocol requirements apply to methods, and three of them are not satisfiable by the released pipelines without changing what is being measured. (i) *Scoring unit.* Their released implementations score conversation-level or agent-level units (or temporal-graph snapshots), whereas every number in Tables 1–3 is computed over partial-order events. Mapping their output onto our events requires a new adapter that defines their scoring unit on V; the resulting rows would report our re-implementation, not the released method, and would not be attributable to the original work. (ii) *Log schema.* Their public inputs do not expose the event fields of Sec. 2 that our testbed logs, and conversely our logs do not contain the trace format their code parses, so neither side can be fed to the other without a lossy conversion whose effect on their score is itself unmeasured. (iii) *Repair interface.* They return an attribution (a failing agent or step) or an agent-level quarantine decision, not an operator applied at an execution element. ASR, ACC, TCR, and cost in Tables 3 and 9 are defined only after an operator is committed, so these columns have no value for such methods unless our own operator set is attached to their output—in which case the recovery numbers would measure our repair module rather than theirs. Where a method additionally requires failure-labeled traces to be tuned or prompted, supplying those labels would also break the benign-only calibration that every row in Table 1 shares (Sec. B.7.2). To keep the comparison aligned while still covering these design principles, the baseline set in Sec. B.5 instantiates one monitor per family under our identical protocol: interaction-graph statistics, topology-guided detection, activation monitoring, trust aggregation, and per-agent sanitization.

B.2 Datasets and Downstream Tasks

All benchmarks use the public MAS workflow templates bundled with each release; we do not change task definitions or agent role graphs relative to those templates. The five workflows are chosen to cover different failure surfaces: pure reasoning chains, debate, retrieval, poisoned external content, and tool execution.

GSM8K (Cobbe et al., 2021). Multi-agent math reasoning (Solver, Critic, Verifier). Downstream metric: **exact match (EM)** on the final numeric answer. Failures propagate mainly along ρ_{msg} edges between solver and critic steps.

CSQA (Talmor et al., 2019). Commonsense multiple-choice debate in the style of Du et al. (2024). Downstream metric: **accuracy (ACC)** on the selected option. Multiple agents argue over the same question, so a single corrupted message can be repeated by several downstream speakers, which is the amplification regime discussed in Appendix D.3.

HotPotQA (Yang et al., 2018). Retriever–reasoner–verifier collaboration on multi-hop questions. Downstream metric: **answer EM / F1** on the short span. Task dependencies are explicit, so L3 coordination edits are meaningful here.

PoisonRAG (Zou et al., 2025). Retrieval pipeline with document poisoning. Downstream metric: **answer correctness (EM/ACC)** under poisoned retrieval. The corruption enters through external content rather than through an agent's own state.

InjecAgent (Zhan et al., 2024). Planner–executor–monitor tool-use MAS. Downstream metric: **task success** and valid tool traces under channel injection. Tool payloads give the message channel a wide benign spread, which makes this the hardest setting for magnitude-only monitors.

B.3 Data Splits and Logging Protocol

Every trajectory is executed once by the unmodified workflow and logged as a partial-order event graph with the parser of Appendix A.1. Logging records agent id, role, task id, state fields, message and tool payloads, and task-progress summaries per event, together with the producer of each consumed message; no agent prompt is modified for instrumentation, so the logged execution is the execution that ran.

Benign train. Fit TRACE-MAS contracts and train or tune baselines that require benign logs; task ids are disjoint from the test split.

Benign validation. Set the threshold τ at quantile level δ and produce the calibration curves of Fig. 5(a–b). No attack labels enter TRACE-MAS training or threshold fitting at any point.

Test. Attacked trajectories for Tables 1–3, with exactly one labeled initiating source c^{gt} per trajectory for localization; three random seeds; mean$\pm$std reported in all tables.

Sec. 4 observations. One corruption object per trajectory; ASR matched across L1–L3 on GSM8K and CSQA so that level-wise detection differences are not confounded by attack strength; the Obs. 3 repair pool uses $n{=}30$ trajectories per hop bucket with clean EM$=$1 and attacked EM$=$0, aligned with Fig. 3.

B.4 Attack Levels (L1–L3)

The three levels differ in *which part of the execution graph* is edited, which is what lets us separate detection difficulty from attack strength.

L1 corrupts agent state (prompt or scratchpad content) without changing wire-level message identity: the sender, receiver, and dependency structure are untouched.

L2 corrupts message or tool payloads while keeping the sender–receiver topology intact, so graph-structure monitors observe an unchanged adjacency.

L3 corrupts coordination itself: dependencies are rewired, roles are skipped, or an agent is taken offline, so the edit is not visible in any single agent's content.

Each test trace carries exactly one attack at the specified level and one ground-truth initiating event, which is what makes Hit@1 well defined. Attack placement is drawn per seed from the eligible events of the trajectory, and the same placement stream is replayed for every defense, so all methods face identical trajectories. On GSM8K and CSQA we additionally match pre-defense ASR across levels (Sec. B.3), so a level-wise drop in detection F1 in Table 1 reflects reduced observability rather than a weaker attack. Ground-truth sources are used only for evaluation; they never enter contract training, threshold calibration, attribution, or repair selection.

B.5 BASELINES

All methods share MAS topology, LLM decoding, attacks, splits, and, where applicable, the same benign calibration rounds $k \in [30, 150]$. Each baseline keeps its native scoring rule and is given the inputs its design requires; we tune its threshold on the same benign validation split we use for τ.

AcMAS(-R) (Xu et al., 2026). Activation and behavior monitoring at agent granularity; the (-R) variant adds generic retry and reassignment as its recovery action. It observes per-agent signals, so L2 and L3 edits that leave agent activations close to benign are only visible through their downstream effects.

BlindGuard (Miao et al., 2026). Interaction-graph anomaly detection from message statistics, without contract-level expectations for a given role and relation type.

G-Safeguard (Wang et al., 2025). Topology-guided detection that assumes a stable workflow graph; it is the most affected by the dynamic topology stress test in Appendix C.1.

A-Trust (He et al., 2025a). Trust aggregation over agents from historical compliance; it scores agents rather than events, so its localization output is an agent, which we map to that agent's highest-scoring event for Hit@1.

AgentSafe (Mao et al., 2025). Per-agent sanitization and self-checks without cross-agent counterfactual rollback.

Oracle Anomaly Ranking. Every row, curve, or legend labeled "Oracle" denotes the **same** weak baseline: *Oracle Anomaly Ranking*, a magnitude and ranking monitor without contract training or counterfactual attribution. It appears in Tables 1–3, Table 9, Fig. 5, the recovery figures, and the Appendix C stress plots whenever the standard baseline set is shown. Mechanism: residual-feature ranking without role-conditioned contract training or counterfactual attribution. It may receive a privileged hint during *benign calibration* for ranking evaluation only, and never receives ground-truth attack locations at test time. Low F1 or suboptimal recovery relative to TRACE-MAS is therefore expected; "Oracle" here is **not** a clairvoyant upper bound.

GT-source repair (diagnostic only). Fig. 3 and related panels use *GT-source repair*, which applies our repair operators at the **ground-truth initiating source** on held-out diagnostic trajectories. It is a reference curve that isolates repair quality from attribution quality; it is **not** Oracle Anomaly Ranking and is **not** available at deployment.

Ablations (Table 4). Independent Contract (per-agent contracts without relation-aware context); w/o Transition Contract (absolute-embedding scoring instead of residual contracts); w/o Violation (no contract alerting, so attribution runs on magnitude ranking); w/o Predictive Repair (fixed operators without rollout-based selection). Each variant is retrained on the same benign split with the same schedule as the full model, so the rows differ only in the removed component.

B.6 METRICS

Unless stated otherwise, every metric is computed per benchmark and per seed and reported as mean$\pm$std over seeds $\{0, 1, 2\}$.

Detection. Trajectory-level F1 and AUROC (Table 1). A trajectory is flagged when its native trajectory score exceeds the method's calibrated threshold; for TRACE-MAS this is $\max_t V(e_t) > \tau$. With TP, FP, and FN counted over trajectories, $\mathrm{F1} = 2\mathrm{TP}/(2\mathrm{TP} + \mathrm{FP} + \mathrm{FN})$. AUROC uses each method's native trajectory score in the orientation declared by that method, with higher meaning more anomalous. The two are reported together because they measure different things and can disagree: F1 is evaluated at each method's calibrated operating point, whereas AUROC is threshold-free and summarizes the ranking over all thresholds.

On AUROC values below 50%. Several baseline entries in Table 1 fall below 50%, and we read them as follows. For a fixed score orientation, AUROC is the probability that a randomly chosen attacked trajectory is ranked above a randomly chosen benign one, and it satisfies $\mathrm{AUROC}(-s) = 1 - \mathrm{AUROC}(s)$ for any score s. A value below 50% therefore states that, on that split, the method's own statistic ranks attacked trajectories *below* benign ones: the score is anti-correlated with the

attack rather than merely uninformative. This is a well-defined outcome of the definition, not a computation error, and it is what the score's declared orientation produces on that split.

Sub-50% AUROC is also reported in prior evaluations of monitors from the same families, including multiple sub-50% entries in the AcMAS evaluation (Xu et al., 2026), so its occurrence here is consistent with published behavior of these scores rather than specific to our testbed.

We do not flip scores post hoc, for two reasons that follow from the protocol. First, choosing the orientation per split requires knowing on that split which direction separates attacks, which is exactly the attack supervision that no method in Table 1 is given (Sec. B.7.2); a deployed monitor must commit to one orientation before the attack is seen. Second, by the identity above, flipping every sub-50% entry would raise all of them above 50% by construction, so the resulting column would report an oracle-selected orientation rather than a property of the detector. Reporting the unflipped value keeps the column comparable across levels and methods.

The sub-50% entries are concentrated at L2 and L3, which is consistent with Obs. 1: when the injected content is carried by messages or dependencies and is then consumed by agents whose own behavior stays on-policy, an agent-centric statistic can move *toward* its benign range on attacked trajectories, so the induced ranking inverts instead of flattening. The very low L3 AUROC of Oracle Anomaly Ranking has a further, structural cause: its trajectory score is a maximum over per-event magnitudes, and L3 attacks that skip a role or take an agent offline remove events, so the attacked trajectory takes a maximum over fewer events and tends to score *below* a benign one. The same rows show that F1 and AUROC need not degrade together, since a method can retain a usable operating point while losing the global ordering.

Localization. Hit@1 and mean hop distance to c^{gt} (Table 2). Hit@1 is the fraction of attacked trajectories whose top-ranked candidate $c^{\star}$ equals the labeled initiating event, and the hop distance is $d(c^{\star}, c^{\mathrm{gt}})$ measured in the partial-order graph, averaged over trajectories. Reporting both matters because a method can miss Hit@1 while staying one hop away (a symptom adjacent to the source) or while pointing at an unrelated branch.

Recovery. ASR and ACC (Table 3); TCR and mean post-alert operator count (Table 9). ASR is the fraction of attacked trajectories in which the attack goal is still achieved after the defense acts, and ACC is the benchmark task metric of Sec. B.2 evaluated on the post-defense trajectory. TCR (task-correct recovery) is computed on the attack-successful pool (clean EM=1, attacked EM=0) and is the fraction of that pool whose task output becomes correct after the committed intervention. Cost is the mean number of post-alert operators applied per trajectory, so a defense that suppresses the attack by editing many events pays for it here rather than in ASR.

Ablation (Table 4). Detect(F1) on **balanced** L1–L3 pooled splits, not level-stratified as in Table 1. Every row, including Full, is retrained end to end with all three seeds (Sec. B.7.3) and evaluated on these pooled splits, so the Full row's Hit@1 and TCR are re-measured under this protocol and are not expected to equal the per-benchmark entries of Tables 2 and 9.

Efficiency. Wall-clock milliseconds for the non-LLM stages, excluding bulk LLM generation (Fig. 6); the corresponding cost model is in Appendix D.6.

B.7 Shared Implementation Settings and Reproducibility

B.7.1 Computation Resources and Software

Table 7 summarizes the machine used for contract training, benign calibration, and non-LLM runtime profiling (Fig. 6). Trajectory generation uses the LLM stack in Table 8; bulk LLM wall time is excluded from the efficiency plots as in Sec. 6, because it is incurred by the underlying MAS rather than by the defense.

We implement TRACE-MAS and the learned baselines in **PyTorch** (CUDA). Event logs are parsed into partial-order graphs with fixed deterministic tie-breaking, so repeated parses of the same log yield identical graphs and identical scores. Frozen text encoding uses **sentence-transformers** (`all-MiniLM-L6-v2`) and is never fine-tuned (Appendix A.3). Multi-agent rollouts call **Qwen2.5-7B-Instruct** (Qwen et al., 2025) through the benchmark GGUF backends (Table 8). Exact Python, PyTorch, and CUDA versions are pinned in the anonymized `requirements.txt`.

B.7.2 UNIFIED EVALUATION PROTOCOL

Following Sec. B.1, all methods in Tables 1, 2, 3, 4, 9, and Fig. 5 share: (i) the same five benchmark workflows and task metrics; (ii) identical L1–L3 injection and disjoint benign train, validation, and test task ids; (iii) the same trajectory LLM decoding (Table 8) for MAS execution and for repair; (iv) the same benign calibration schedule ($k \in [30, 150]$ rounds with shared quantile refresh); (v) three random seeds on attacked test trajectories (mean$\pm$std). TRACE-MAS fits contracts on the benign train split only, and attack labels never enter training or threshold fitting. ExecFormer hyperparameters are fixed in Appendix A (Table 6) and are not re-tuned per benchmark, so no part of the reported gain comes from per-dataset search.

B.7.3 RANDOMNESS AND REPORTING

Seeds $\{0, 1, 2\}$ control attack placement, dataloader order, and dropout where present. Contract training initializes with seed 0 unless an ablation retrains end to end (Table 4), in which case all three seeds are used per variant. No post-hoc score flipping is applied anywhere, including AUROC (Sec. B.6).

Calibration (Q4). Each calibration round refreshes the benign quantile on the validation split; Fig. 5 uses the same schedule for every method, so the curves compare data efficiency of calibration rather than different calibration budgets.

Deployment stress (Appendix C). Message delay $\ell \in \{0, \ldots, 5\}$; agent count $N \in \{2, 4, 8, 16, 32\}$; dynamic-edge fraction in $[0.2, 1.0]$; execution imbalance $\mathrm{IR} = \min_i f_i / \max_i f_i$ over per-agent event counts, in $[0.2, 1.0]$. One factor is varied at a time with attacks, splits, and calibration protocol held fixed.

B.7.4 SUPPLEMENTARY TABLES (HARDWARE, LLM, RECOVERY COST)

Tables 7–9 collect the implementation settings and the Q3 supplement referenced above. Table 9 reports TCR↑ and post-alert operator cost (Cost↓) alongside Table 3; per-column best results are **bold**. Reading the two columns together is the point of the table: a method can raise TCR by intervening aggressively, so the cost column is what distinguishes recovery from over-correction.

Table 7: Hardware used for contract training, calibration, and non-LLM profiling.

Component	Specification (measured)
CPU	2× Intel Xeon Gold 6530 (128 threads)
RAM	503 GiB
GPU	8× NVIDIA RTX A6000 (∼48 GB each)

Table 8: Trajectory LLM and decoding defaults shared across benchmarks.

Item	Setting
Backbone	Qwen2.5-7B-Instruct (all five benchmarks)
Format / runtime	GGUF inference; 40 layers on GPU
Max new tokens / step	128
Decoding	Benchmark template defaults (temperature, top-p) unless noted in release

Table 9: Recovery quality and intervention cost (Q3 supplement to Table 3). TCR↑ and post-alert operator count Cost↓ on five benchmarks (mean±std, %). Best **bold**; second-best underlined.

Dataset	GSM8K		CSQA		HotPotQA		InjecAgent		PoisonRAG	
Method	TCR↑	Cost↓	TCR↑	Cost↓	TCR↑	Cost↓	TCR↑	Cost↓	TCR↑	Cost↓
Oracle	$10.67_{\pm 0.96}$	$10.84_{\pm 0.25}$	$18.67_{\pm 0.69}$	$12.05_{\pm 0.24}$	$8.00_{\pm 0.98}$	$12.55_{\pm 0.27}$	$13.33_{\pm 0.77}$	$13.04_{\pm 0.44}$	$6.67_{\pm 0.16}$	$13.89_{\pm 0.28}$
BlindGuard	$14.67_{\pm 0.95}$	$5.93_{\pm 1.30}$	$13.33_{\pm 0.67}$	$5.45_{\pm 1.68}$	$2.67_{\pm 0.65}$	$5.82_{\pm 1.52}$	$26.67_{\pm 0.28}$	$5.12_{\pm 0.49}$	$5.33_{\pm 0.58}$	$5.30_{\pm 0.88}$
G-Safeguard	$12.00_{\pm 0.30}$	$10.07_{\pm 0.73}$	$9.33_{\pm 0.60}$	$8.49_{\pm 0.47}$	$5.33_{\pm 0.58}$	$10.88_{\pm 1.54}$	$20.00_{\pm 1.04}$	$9.67_{\pm 1.21}$	$2.67_{\pm 0.61}$	$10.91_{\pm 0.96}$
AcMAS-R	$20.41_{\pm 0.25}$	$1.26_{\pm 0.64}$	$23.33_{\pm 0.47}$	$1.22_{\pm 0.04}$	$16.27_{\pm 0.38}$	$2.82_{\pm 0.14}$	$19.67_{\pm 0.58}$	$2.43_{\pm 0.04}$	$22.78_{\pm 0.78}$	$1.22_{\pm 0.05}$
A-Trust	$34.67_{\pm 0.95}$	$12.78_{\pm 0.96}$	$30.67_{\pm 0.96}$	$12.86_{\pm 0.38}$	$9.33_{\pm 0.16}$	$10.77_{\pm 0.96}$	$45.33_{\pm 1.07}$	$12.38_{\pm 0.72}$	$16.00_{\pm 0.55}$	$12.89_{\pm 0.92}$
AgentSafe	$38.67_{\pm 0.20}$	$9.56_{\pm 0.63}$	$40.00_{\pm 0.43}$	$11.57_{\pm 0.41}$	$13.33_{\pm 0.07}$	$8.79_{\pm 0.65}$	$33.33_{\pm 0.16}$	$9.59_{\pm 0.54}$	$13.33_{\pm 0.67}$	$9.92_{\pm 0.47}$
Ours	**96.00**$_{\pm 0.18}$	**0.96**$_{\pm 0.52}$	**76.67**$_{\pm 0.94}$	**1.06**$_{\pm 0.13}$	**81.58**$_{\pm 0.12}$	**0.89**$_{\pm 0.08}$	**73.77**$_{\pm 0.49}$	**0.53**$_{\pm 0.04}$	**82.90**$_{\pm 0.39}$	**0.54**$_{\pm 0.06}$

C Additional Results

Sec. 6.5 reports benign calibration and hyperparameter sensitivity in the main text (Fig. 5). This appendix reports the two experiment groups deferred from there: (i) deployment robustness, i.e. whether contracts calibrated on benign logs remain useful when the runtime environment differs from the one assumed by the workflow template (Sec. C.1); and (ii) localization under different MAS communication structures (Sec. C.2). It then reports two checks of the repair stage: (iii) how faithfully the contract-space rollout of Appendix A.7.2 predicts real re-execution (Sec. C.3); and (iv) a comparison of localization and repair under the same input, the same operators, and the same budget (Sec. C.4).

Protocol (Secs. C.1–C.2). All stress tests vary **one** factor at a time. Attacks, task splits, seeds, and the metric definitions are those of Appendix B, and the benign calibration procedure of Appendix B.7.2 is applied unchanged in every configuration, so no method is evaluated with a threshold fitted under a different regime. Method-side hyperparameters stay at the defaults of Appendix A.9; nothing is re-tuned per configuration. Each panel therefore isolates the effect of the environment on the monitors themselves rather than on their tuning.

C.1 Deployment Robustness

The four factors below correspond to assumptions that monitoring methods commonly make about an MAS: synchronous visibility, fixed scale, static topology, and comparable per-agent activity. Each factor breaks one of them.

Message delay (Fig. 7). End-to-end message visibility is delayed by $\ell \in \{0, \ldots, 5\}$ units, so an event may be observed well after the step that produced it. TRACE-MAS keeps detection F1 stable because the partial-order event graph is built from producer–consumer links rather than from wall-clock rounds (Appendix A.1): delaying delivery changes when an edge is observed but not which events it connects, and the context c_t the contract conditions on is unchanged. Timing-sensitive baselines (AcMAS(-R), A-Trust, AgentSafe) degrade as delay grows, since their per-round aggregates mix events that no longer belong to the same logical step.

Agent scale (Fig. 8). We replicate role templates for $N \in \{2, 4, 8, 16, 32\}$, keeping the task and attack protocol fixed. TRACE-MAS remains comparatively flat because scoring is per event and conditioned on role and relation type, so adding agents adds events without diluting the signal at the corrupted one. Trust- and activation-heavy monitors lose signal as graphs grow, since they aggregate over more agents to reach one decision and the corrupted contribution shrinks in the pooled statistic.

Dynamic topology (Fig. 9). We vary the fraction of time-varying communication edges in $[0.2, 1.0]$. Contract scores do not reference a fixed adjacency matrix: the encoder aggregates whichever parents an event actually has, so a re-routed message is still scored against the contract for its relation type. Topology-centric baselines drop when edges churn, because edge churn is itself the anomaly signal they rely on.

Execution imbalance (Fig. 10). The per-agent event-rate ratio $\mathrm{IR} = \min_i f_i / \max_i f_i \in [0.2, 1.0]$ (1 is balanced; smaller is more skewed) sets how unevenly the agents speak. Because the contract predicts a context-dependent spread Σ_t as well as a mean (Appendix A.4), a talkative agent's larger raw deviations are normalized by its own benign variability, which is exactly the effect described

by Property 3 in Appendix D.2. Role-conditioned residuals therefore avoid the volume-only false alarms that hurt uniform-rhythm heuristics.

Taken together, the four panels test the deployment assumption behind Q4: thresholds fit on benign logs should stay usable when coordination timing, scale, and structure move away from the calibration setting, and the factors that degrade the baselines are the ones their representations depend on.

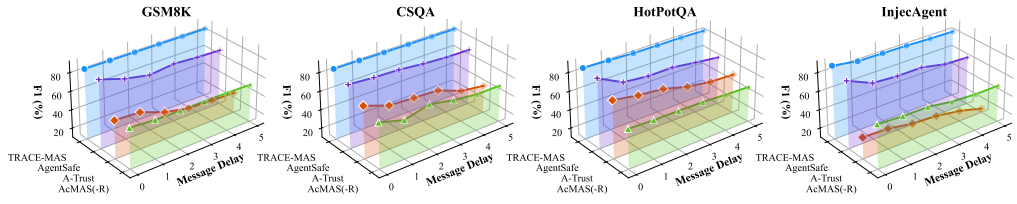


Figure 7: Effect of alert delay. Detection F1 under increasing end-to-end latency on GSM8K, CSQA, HotPotQA, and InjecAgent. TRACE-MAS does not assume synchronous communication and stays comparatively flat.

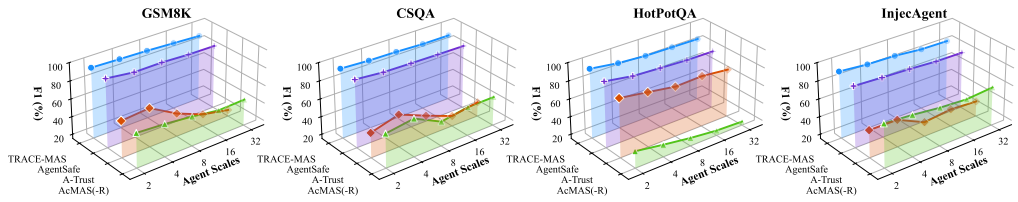


Figure 8: Robustness to agent scale. Detection F1 vs. $N \in \{2, 4, 8, 16, 32\}$ on four benchmarks.

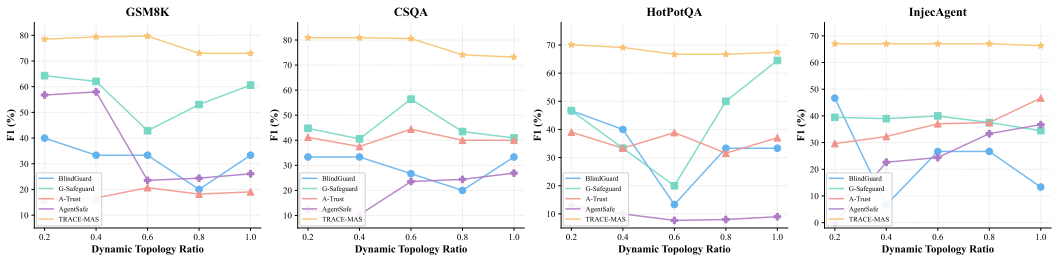


Figure 9: Robustness to dynamic topology. Detection F1 vs. dynamic-edge fraction on four benchmarks.

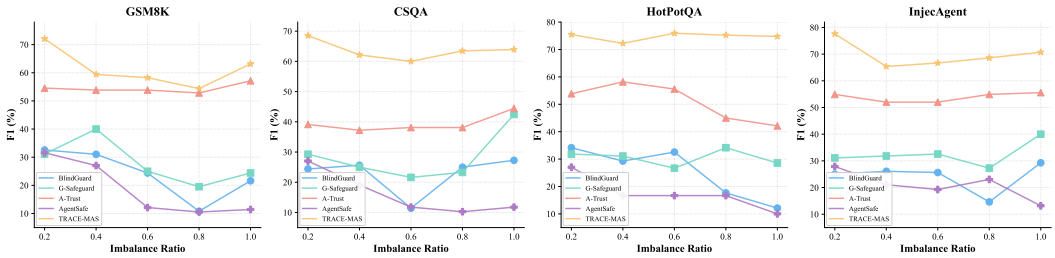


Figure 10: Robustness to execution imbalance. Detection F1 vs. imbalance ratio IR on four benchmarks.

C.2 Localization under Varying Topologies

Detection robustness does not imply attribution robustness: changing the communication structure changes the ancestor sets that candidate sources are drawn from, and hence the number of plausible explanations per alert. We therefore evaluate Hit@1 on GSM8K, CSQA, HotPotQA, and PoisonRAG under Chain, Star, Hierarchical, and Dense communication templates (Fig. 11), with attacks, splits, and defaults unchanged.

The four templates span the range that matters for attribution. A Chain gives each alert a single upstream path, so candidates are few and mostly on one branch; a Star concentrates traffic through a hub whose events are ancestors of almost everything; a Hierarchy adds depth, which increases the hop distance between an injected source and the event where it becomes visible; and a Dense template maximizes the number of ancestors within the hop limit H, so magnitude-based ranking has the most opportunity to pick a symptom.

TRACE-MAS consistently outperforms AgentSafe, G-Safeguard, and A-Trust across these structures because attribution scores the violation drop obtained by restoring a candidate rather than the violation level of an event (Appendix A.6.3). By the locality of Restore (Property 4 in Appendix D.4), candidates that are not upstream of the corrupted branch contribute no drop and are ranked out automatically, which is what keeps the candidate set manageable in the Star and Dense templates where the raw ancestor count is largest.

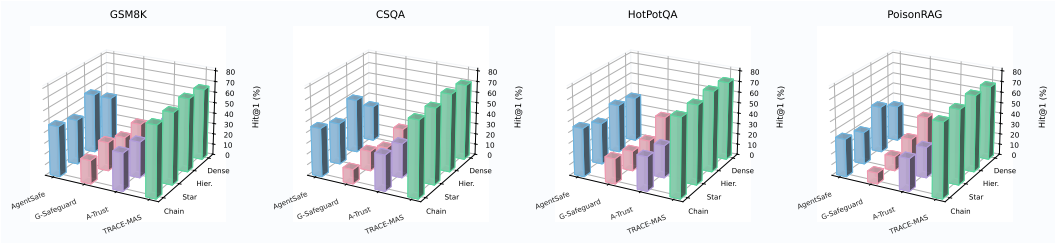


Figure 11: Localization under varying MAS topologies. Hit@1 on four benchmarks over Chain, Star, Hierarchical, and Dense structures.

C.3 Rollout Fidelity under Real Re-execution

Repair selection ranks the operators in $\mathcal{U}(c^\star)$ by the rollout estimate $\hat{\mathcal{J}}(u)$ instead of executing each of them. This subsection tests that substitution directly by executing every candidate.

Paired pool. Secs. C.3 and C.4 share one *paired ablation pool* $\mathcal{P}$: attack-successful trajectories from the Q3 repair pool (Sec. B.6), with $|\mathcal{P}|$=6 trajectories, the **E2-lite** commit path, i.e. the repair stage of Sec. A.7 run at a budget of B=1 committed operator per trajectory with LLM regeneration restricted to the edited branch, the deployment defaults K=1 and Appendix A.9, and seed 0. Because both subsections need every candidate operator re-executed by the trajectory LLM, $\mathcal{P}$ is sized for exhaustive re-execution rather than for the per-benchmark resolution of the main tables, and all numbers below are single-seed. This pool is smaller than the full attacked-test splits behind

Table 2 and is used only to pair localization, rollout, and re-execution under identical inputs; it is **not** a re-report of Table 2.

Procedure. For each trajectory in $\mathcal{P}$ the attributed event $c^\star$ is fixed. For every $u \in \mathcal{U}(c^\star)$ we compute (i) $\hat{\mathcal{J}}(u)$ from Appendix A.7.2 and (ii) $\mathcal{J}(u)$ after committing u and re-executing $\mathrm{Desc}(c^\star)$ with the same decoding and template steps. Every candidate is executed once offline; deployment still commits only $u^\star = \arg\min_u \hat{\mathcal{J}}(u)$.

Reported quantities. *Match* is the fraction of trajectories on which re-execution under every u yields the same K window events as the rollout (Property 7 precondition). *Agr* is the fraction with $u^\star = u^{\mathrm{opt}}$ for $u^{\mathrm{opt}} = \arg\min_u \mathcal{J}(u)$. Spearman ρ is the rank correlation between $\hat{\mathcal{J}}$ and $\mathcal{J}$ over $\mathcal{U}(c^\star)$, averaged over trajectories; it measures agreement of the induced *ordering*, which is all that selection uses, and not the size of $|\hat{\mathcal{J}} - \mathcal{J}|$. Once $u^\star = u^{\mathrm{opt}}$ on every trajectory, the regret of Property 6 and ΔTCR are zero by definition.

Table 10: Rollout fidelity under real LLM re-execution on the paired ablation pool (offline exhaustive re-execution per candidate; deployed cost remains one commit).

Metric (paired pool $\mathcal{P}$, K=1, E2-lite, B=1)	Value
Match: rollout window = re-execution window (%) ↑	100
Agr: $u^\star = u^{\mathrm{opt}}$ (%) ↑	100
Spearman$(\hat{\mathcal{J}}, \mathcal{J})$ ↑	1.00
ΔTCR: TCR($u^\star$)−TCR(u^{opt}) (points)	0

Three readings follow, and they are deliberately limited to what the table measures. First, the precondition of Property 7 holds throughout: at K=1 no operator in $\mathcal{U}(c^\star)$ added or removed a window event under re-execution, so the matched-window form of Eq. (56) applies on every trajectory. Second, the rollout orders the operators exactly as re-execution does (ρ=1.00) and selects the same operator on every trajectory, so the conclusion of Property 6 and Corollary 1 holds here with zero regret, and the task cost of ranking by $\hat{\mathcal{J}}$ instead of executing all candidates is zero. Third, this is evidence about the *ordering* rather than about the magnitude of $\nu(u)$ and $\theta(u)$: a rank correlation of 1.00 is compatible with a non-zero value gap between $\hat{\mathcal{J}}$ and $\mathcal{J}$, and what the result shows is that whatever gap remains is smaller than the separation between operators, which is the condition Corollary 1 requires. Since the deployed rule uses only the ordering, this is the property the repair stage depends on. Both tables are measured on $\mathcal{P}$; the Q3 numbers in Tables 3 and 9 remain those of the full test splits.

C.4 Localization and Repair under the Same Input, Operators, and Budget

Tables 3 and 9 evaluate each defense with its native response (retry, quarantine, sanitization, or our repair), so they compare complete defenses. This subsection separates the two stages on the paired pool $\mathcal{P}$ of Sec. C.3.

Protocol. *Same input:* every row localizes on the same six trajectories in $\mathcal{P}$, whether or not its own detector would have fired. *Same operators:* each row's top-ranked event is passed to the identical E2-lite repair module ($\mathcal{U}(\cdot)$, rollout with K=1, λ_1, λ_2 as in Sec. A.7). *Same budget:* B=1 committed operator per trajectory and the same regeneration scope. Hit@1 is the fraction with $c^\star = c^{\mathrm{gt}}$ on $\mathcal{P}$ only; because $|\mathcal{P}|$=6, percentages appear in steps of $100/6 \approx 16.7$ and **must not be compared** to Table 2, which aggregates the full attacked-test protocol.

Localizers (Table 11). Each baseline supplies its own ranking; *Full monitoring* is contract violation alerting with $\arg\max_t V(e_t)$ localization, i.e. the "w/o CVA" diagnostic of Fig. 2, in which counterfactual violation attribution (CVA) is replaced by max-violation ranking, so neither $\mathrm{Restore}$ nor $R(c)$ is used; TRACE-MAS *(CVA)* is the full rule $c^\star = \arg\max_c R(c)$ of Sec. 5.3. TCR uses the definition of Sec. B.6, evaluated on $\mathcal{P}$, whose trajectories are attack-successful by construction.

Selection rules (Table 12). With $c^\star$ fixed, we vary only how u is chosen over $\mathcal{U}(c^\star)$: the fixed operator used in the w/o Predictive Repair ablation, a uniform random operator, rollout rank-

ing ($\arg\min_u \hat{\mathcal{J}}(u)$), and exhaustive re-execution ranking ($\arg\min_u \mathcal{J}(u)$, the reference from Sec. C.3).

Table 11: Paired ablation on pool $\mathcal{P}$ ($|\mathcal{P}|$=6): localizer varied, repair module and budget fixed (Hit@1 and TCR in %, single seed). One trajectory is 16.7 points, so single-trajectory gaps are within the resolution of the pool and no entry is bolded; the paper's method comparisons are Tables 2 and 9, to which these values are not comparable.

Localization source (same E2-lite repair, B=1)	Hit@1↑	TCR↑
TRACE-MAS (CVA)	16.7	66.7
Full monitoring (Max-V)	33.3	33.3
AcMAS	16.7	16.7
AgentSafe	33.3	50.0
G-Safeguard	0.0	33.3
A-Trust	33.3	33.3

Table 12: Paired ablation on $\mathcal{P}$: selection rule varied (TCR in %, single seed). Rollout selection commits the same operator as exhaustive re-execution, consistent with Table 10.

Selection over $\mathcal{U}(c^\star)$ ($c^\star$ fixed, E2-lite, B=1)	TCR↑
Rollout $\hat{\mathcal{J}}$, K=1 (ours)	66.7
Fixed operator	50.0
Random operator	33.3
Re-execution $\mathcal{J}$ (ref.)	66.7

On $\mathcal{P}$, sharing the repair module isolates *where* to act from *how* to act. The pool is small enough that a one-trajectory difference (16.7 points) is within its resolution, so we read only the larger gaps. Read with that caveat, responsibility attribution recovers the most tasks of any localizer in the table: two trajectories more than max-violation ranking (66.7 vs. 33.3) and than the trust- and topology-centric localizers G-Safeguard and A-Trust, three more than the activation-centric AcMAS (16.7), and one more than AgentSafe (50.0). The comparison with Max-V is the informative one, because that row shares our violation score and our repair module and differs only in replacing $R(c)$ with $\arg\max_t V(e_t)$; the gap is the direction Obs. 2 and Property 5 predict, namely that the event with the largest violation is not the event whose repair restores the task. Two entries should not be over-read. Max-V is one trajectory above CVA on Hit@1 here, which this pool cannot resolve and which the full attacked-test protocol of Table 2 settles in the other direction; and the one-trajectory gap to AgentSafe is at the resolution of the pool. What the pool adds over Table 9 is that Hit@1 and TCR are measured on identical inputs through an identical repair module at an identical budget, so a difference between rows is attributable to the localization stage alone; it is not itself a method ranking, and the paper's recovery claims remain those of Tables 3 and 9.

Two features of the table follow from the pool rather than from the methods and are worth stating so they are not read as inconsistencies. First, every Hit@1 here is far below the corresponding column of Table 2: $\mathcal{P}$ contains only attack-successful trajectories at a single seed, i.e. the cases in which the corruption defeated the undefended workflow, and six such trajectories are not a localization measurement. This is why the localization claims of the paper are made on the full protocol and why we read only TCR differences here. Second, TCR exceeds Hit@1 for every row, which is expected rather than contradictory: recovery does not require the exact initiating event, since committing an operator on the corrupted branch near the source already restores many tasks, the effect quantified by the hop buckets of Fig. 3 and by the tolerance to bounded amplification in Property 5. Hit@1 and TCR therefore answer different questions on the same inputs, which is what makes the paired design useful.

Table 12 isolates the selection rule at a fixed $c^\star$, so its first row is the same configuration as the TRACE-MAS (CVA) row of Table 11 and carries the same value by construction. Rollout selection equals exhaustive re-execution, which is forced by Table 10: at Agr $= 100\%$ the two rules commit the same operator on every trajectory, so they cannot differ in TCR. It is one trajectory above the

fixed operator and two above random selection, in the direction of the w/o Predictive Repair ablation, whose effect is quantified on the full splits in Table 4. The point of the table is the pair of equalities and inequalities together: choosing the operator by $\hat{\mathcal{J}}$ costs no task recovery relative to executing all candidates, while ranking by anything cheaper than the rollout does.

D ANALYTICAL SUPPORT

This appendix explains, at the level of the quantities TRACE-MAS computes, why the three stages of Sec. 5 are built the way they are. Sec. D.2 shows that the violation score of Sec. 5.2 is the training objective of that section evaluated at test time, that the benign quantile threshold of Eq. (30) controls false alarms without attack labels, and that the detectable margin of a corruption is exactly the Mahalanobis norm of the deviation it induces. Sec. D.3 derives why ranking events by that margin does not localize the source. Sec. D.4 shows how the locality of $\mathrm{Restore}$ makes the responsibility score of Sec. 5.3 separate an initiating event from its symptoms, and states the one behavioral condition under which the separation holds. Sec. D.5 gives the stability of the repair-selection rule of Sec. 5.4 under rollout error and bounds that error for the rollout of Appendix A.7.2 in terms of the calibrated threshold τ and the depth K, and Sec. D.6 accounts for the cost of the pipeline. Sec. D.7 states what is not claimed.

All statements below are properties of the model we fit and the scores we report; none of them is used to substitute for the empirical results in Sec. 6.

D.1 SETUP AND NOTATION BRIDGE

We reuse the notation of Appendix A. For event e_t with encoded context c_t, the contract is the Gaussian $p_\theta(\Delta e_t \mid c_t) = \mathcal{N}(\mu_t, \Sigma_t)$ with diagonal $\Sigma_t \succ 0$ (Eq. (26)), and the violation score is the Mahalanobis deviation $V(e_t)$ of Eq. (28) between the observed transition $\widetilde{\Delta e}_t$ and μ_t. Since the channel weights (β, γ) of Eq. (28) are equivalent to the rescaled diagonal covariance $\tilde{\Sigma}_t = \mathrm{diag}(\Sigma_t^h, \Sigma_t^m/\beta, \Sigma_t^s/\gamma)$, we keep Σ_t for the trained covariance and write $\tilde{\Sigma}_t$ for the scoring covariance, so that $V(e_t) = \|\widetilde{\Delta e}_t - \mu_t\|^2_{\tilde{\Sigma}_t^{-1}}$; the two coincide at $\beta = \gamma = 1$. With d_h, d_m, d_s the channel dimensions ($d_z = d_h + d_m + d_s$), we write $\bar{d} = \mathrm{tr}(\tilde{\Sigma}_t^{-1}\Sigma_t) = d_h + \beta d_m + \gamma d_s$, which equals d_z at $\beta = \gamma = 1$. The residual shrinkage of Appendix A.3 multiplies every violation term by the same constant γ^2_{res}. This changes no alert and not Ω, and rescales the margins of Property 3; in $R(c)$ and in the repair objective it rescales the violation terms relative to λ_{pen}, λ_1, and λ_2, so the statements below, written for the unshrunk score, hold with these weights read as $\lambda/\gamma^2_{\mathrm{res}}$. We write $\|b\|^2_A = b^\top A\, b$, d_z for the dimension of Δe_t, $d(c, e)$ for the hop distance in the partial-order graph, $\mathrm{Desc}(c)$ for the strict descendants of c (excluding c), and

$$\Delta_c(e) \;=\; V(e) - V_c^{\mathrm{res}}(e) \tag{43}$$

for the violation drop at e after restoring candidate c, where the sum in Eq. (35) runs over c together with its descendants. Because $\mathrm{Restore}_c$ writes the contract mean μ_c while holding the context c_c at its pre-restoration value (Appendix A.6.2), $V_c^{\mathrm{res}}(c) = 0$ and hence $\Delta_c(c) = V(c)$. Since $V(e) - \tilde{V}_c(e) = \omega\, \Delta_c(e)$, the responsibility score separates into a counterfactual term and the validity penalty,

$$R(c) \;=\; \omega\, R_{\mathrm{cf}}(c) \;-\; \lambda_{\mathrm{pen}}\, \mathcal{P}(c), \qquad R_{\mathrm{cf}}(c) \;=\; \sum_{e \in \{c\} \cup \mathrm{Desc}(c)} \psi^{\,d(c,e)}\, \Delta_c(e), \tag{44}$$

so the gate ω only rescales R_{cf} against $\mathcal{P}$. All comparisons below are statements about R_{cf} and about the penalty difference between two candidates, so they hold for any $\omega > 0$.

Throughout, a corruption at event e is described by the deviation it induces in contract space: the realized residual becomes

$$\widetilde{\Delta e}_t \;=\; \mu_t + b_t + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \Sigma_t), \tag{45}$$

where b_t is the induced shift ($b_t = 0$ on benign events) and ε_t is the benign variability the contract already accounts for. This description is not an assumption about the attacker: any edit to a logged field produces some b_t, and b_t is exactly the object our score sees. It does assume that the contract is well specified on benign data, i.e. that benign residuals follow $\mathcal{N}(0, \Sigma_t)$ with the trained Σ_t; Property 3 and Sec. D.3 rely on this, whereas Properties 1, 2, 4, and 6 do not, and Property 7(b) uses instead the exchangeability assumption stated there.

D.2 From Benign Training to the Violation Score

Property 1 (the score is the training objective). For the scoring Gaussian $\mathcal{N}(\mu_t, \tilde{\Sigma}_t)$,

$$-\log \mathcal{N}(\widetilde{\Delta e}_t; \mu_t, \tilde{\Sigma}_t) \;=\; \tfrac{1}{2}\Big[V(e_t) + \log\det \tilde{\Sigma}_t + d_z \log 2\pi\Big], \tag{46}$$

so $V(e_t) = -2\log \mathcal{N}(\widetilde{\Delta e}_t; \mu_t, \tilde{\Sigma}_t) - \log\det \tilde{\Sigma}_t - d_z \log 2\pi$.

Support. Substitute $\mathcal{N}(\mu_t, \tilde{\Sigma}_t)$ into the Gaussian density; since $\tilde{\Sigma}_t^{-1} = \mathrm{diag}\big((\Sigma_t^h)^{-1}, \beta(\Sigma_t^m)^{-1}, \gamma(\Sigma_t^s)^{-1}\big)$, the quadratic form is $V_h + \beta V_m + \gamma V_s$, i.e. Eq. (28). □

Property 1 is an identity, not an approximation. At $\beta = \gamma = 1$ it states that detection does not introduce a new criterion at test time: the score is the quadratic term of the negative log-likelihood in Eq. (27), evaluated on the test transition. With $(\beta, \gamma) = (1, 0.5)$ the scored quantity is the same likelihood with the completion channel's variance doubled, i.e. a fixed reweighting of the trained contract rather than a different model. The implemented training loss additionally weights the per-channel likelihoods by $(\lambda_h, \lambda_m, \lambda_s)$ and adds two regularizers (Appendix A.4); the identity therefore ties the score to the Gaussian model that training fits (the same μ_t and Σ_t), not to the exact value of the implemented loss. It also explains why the spread must be predicted rather than fixed: normalizing by $\tilde{\Sigma}_t^{-1}$ is what makes the residual comparable across roles with different benign variability, and the $\log\det$ term of the training likelihood is what prevents the learned spread from growing to absorb large residuals.

Property 2 (benign calibration controls false alarms). Let $V_1, \ldots, V_n$ be the violation scores of the benign validation events and let V^* be the score of a fresh benign event, and suppose the calibration and test benign events are exchangeable and the score function is fixed before calibration (the contract is fit on the disjoint training split). If τ is the empirical δ-quantile of $\{V_i\}$ as in Eq. (30) with α=1, then

$$\Pr\big[V^* > \tau\big] \;\le\; 1 - \delta + \tfrac{1}{n+1}, \tag{47}$$

and for a benign trajectory with $|\mathcal{V}|$ events the alarm probability obeys

$$\Pr\big[\exists\, t:\; V(e_t) > \tau\big] \;\le\; |\mathcal{V}|\,\big(1 - \delta + \tfrac{1}{n+1}\big). \tag{48}$$

Support. The first inequality is the standard exchangeability (conformal) bound (Vovk et al., 2005; Angelopoulos & Bates, 2023) for an empirical quantile threshold; the second follows by a union bound over the events of the trajectory, which requires no independence between events. □

Property 2 is the reason the calibration protocol of Sec. A.5 needs no attack labels: the nominal false-alarm level is a property of δ and the benign calibration sample, so τ can be fixed before any attack is observed. It also quantifies the choice δ=0.99, since the trajectory-level bound degrades linearly with trajectory length and is informative only when $|\mathcal{V}|(1-\delta)$ is small; for long trajectories it is a worst-case statement rather than a prediction of the observed alarm rate. Raising the threshold to $\alpha \cdot \mathrm{Quantile}_\delta$ with $\alpha > 1$ can only lower the alarm probability, so the bound continues to hold, whereas $\alpha < 1$ is not covered by it. The bound is one-sided by design; it says nothing about detection power, which is the content of Property 3 and of Sec. 6.

Property 3 (detectable margin of a corruption). Under the description of Eq. (45),

$$\mathbb{E}\big[V(e_t)\big] \;=\; \bar{d} + \|b_t\|^2_{\tilde{\Sigma}_t^{-1}}, \tag{49}$$

so a benign event has $\mathbb{E}[V] = \bar{d}$ and the expected margin of a corrupted event over a benign one is exactly $\|b_t\|^2_{\tilde{\Sigma}_t^{-1}}$. At $\beta = \gamma = 1$ this reads $\mathbb{E}[V] = d_z + \|b_t\|^2_{\Sigma_t^{-1}}$, and V is non-central $\chi^2_{d_z}$ with $\mathrm{Var}[V] = 2d_z + 4\|b_t\|^2_{\Sigma_t^{-1}}$.

Support. With $x = b_t + \varepsilon_t$ and $\varepsilon_t \sim \mathcal{N}(0, \Sigma_t)$, $\mathbb{E}[x^\top \tilde{\Sigma}_t^{-1} x] = \mathrm{tr}(\tilde{\Sigma}_t^{-1}\Sigma_t) + b_t^\top \tilde{\Sigma}_t^{-1} b_t$. At $\beta = \gamma = 1$, $\Sigma_t^{-1/2} x \sim \mathcal{N}(\Sigma_t^{-1/2} b_t, I)$, and the variance is that of a non-central χ^2 with d_z degrees of freedom and non-centrality $\|b_t\|^2_{\Sigma_t^{-1}}$. □

Property 3 turns a design question into a measurable one. Whether a corruption is visible depends not on its textual size but on the Mahalanobis norm of the deviation it induces *relative to the benign spread of that context*, the same quantity that Mahalanobis-based out-of-distribution detectors threshold (Lee et al., 2018). This has three consequences for our design. First, conditioning the contract on role and relation type (Sec. A.2) shrinks Σ_t for a given edit and therefore increases $\|b_t\|^2_{\tilde{\Sigma}_t^{-1}}$: the same injected message is more visible against a role-specific expectation than against a global one. Second, modeling transitions rather than absolute embeddings (Sec. A.3) removes benign content variation that would otherwise inflate Σ_t. Third, it explains Obs. 1 without assuming anything about the attacker: a coordination-level (L3) edit can leave the state channel almost unchanged, so monitors restricted to agent activations see a small b_t in the only channel they observe, while the same edit produces a large shift in the message and completion channels the contract covers. We therefore do not assume a separation margin between attacked and benign transitions; Property 3 identifies what such a margin would consist of, and Table 1 measures whether it materializes.

D.3 Why Magnitude Ranking Does Not Localize

Detection asks whether some event deviates; attribution asks which event started it. These are different questions because the deviation propagates.

Consider a corruption at c^{gt} and a descendant v reached along $E_{\mathrm{msg}} \cup E_{\mathrm{dep}}$. The context c_v that the contract conditions on is computed from the (now corrupted) fields of the events upstream of v, so both the observed residual at v and the prediction μ_v change. Writing the induced shift at v to first order in the shift at the source,

$$b_v \;\approx\; J_{v\leftarrow c^{\mathrm{gt}}}\, b_{c^{\mathrm{gt}}}, \qquad J_{v\leftarrow c^{\mathrm{gt}}} \;=\; \frac{\partial\big(\widetilde{\Delta e}_v - \mu_v\big)}{\partial b_{c^{\mathrm{gt}}}}, \tag{50}$$

Property 3 gives the resulting margins, and their ratio is

$$\frac{\mathbb{E}[V(v)] - \bar{d}}{\mathbb{E}[V(c^{\mathrm{gt}})] - \bar{d}} \;\approx\; \frac{\|J_{v\leftarrow c^{\mathrm{gt}}} b_{c^{\mathrm{gt}}}\|^2_{\tilde{\Sigma}_v^{-1}}}{\|b_{c^{\mathrm{gt}}}\|^2_{\tilde{\Sigma}^{-1}_{c^{\mathrm{gt}}}}}, \tag{51}$$

where the approximation is only the first-order one above. Nothing forces this ratio to be smaller than one. It exceeds one whenever the propagation operator amplifies in the $\tilde{\Sigma}$-metric, which happens in exactly the situations our benchmarks contain: a downstream role with tighter benign spread than the injecting role (small Σ_v), or a corrupted payload that a downstream agent expands into a longer, more confidently expected step. This is the formal content of Obs. 2 and of the mean-hop column of Table 2: the largest-violation event is frequently a symptom rather than the origin, so $\arg\max_t V(e_t)$ is not guaranteed to return c^{gt}, the same gap that motivates counterfactual root-cause analysis of outliers in causal models (Budhathoki et al., 2022). Throughout Secs. D.3–D.4, c^{gt} is the true initiating event, and the question is whether $c^\star = \arg\max_c R(c)$ recovers it.

We stress that we do *not* posit an attenuating-propagation condition of the form $V(v) \le r^{d(c^{\mathrm{gt}},v)} V(c^{\mathrm{gt}})$ with $r < 1$. Such a condition would make magnitude ranking correct by fiat and would contradict our own measurements. The attribution rule of Sec. 5.3 is designed for the regime in which the ratio above can be large, and the next subsection shows why it survives there.

D.4 Why Counterfactual Responsibility Separates Source from Symptom

Two facts about $\mathrm{Restore}$ carry the argument; only the second involves the behavior of the learned contract.

Property 4 (locality of restoration). $\mathrm{Restore}(\tilde{G}, c, \cdot)$ modifies the fields of c and the contexts of $\mathrm{Desc}(c)$ only. Hence $\Delta_c(e) = 0$ for every $e \notin \{c\} \cup \mathrm{Desc}(c)$, and $R_{\mathrm{cf}}(c)$ is a sum over $\{c\} \cup \mathrm{Desc}(c)$ only.

Support. Immediate from the definition of $\mathrm{Restore}$ (Sec. A.6.2) and from the encoder being causal: the context of an event depends only on that event and its ancestors. □

Property 4 already removes a large class of candidates from contention. If c is unrelated to the corrupted branch, then no corrupted event lies in $\{c\} \cup \mathrm{Desc}(c)$, so $R_{\mathrm{cf}}(c)$ collects only its own term

$\Delta_c(c) = V(c)$ and benign-level changes on its descendants, all at the scale of benign variability; such a candidate cannot outrank one that explains real violation mass except through the validity penalty. The remaining competition is therefore between the true source and the symptoms on its own branch, and that is where the hop weights and the source's own term act.

Condition 1 (restoration at the origin is at least as effective). Let $v \in \mathrm{Desc}(c^{\mathrm{gt}})$ be a symptom such that every path from c^{gt} to an event of $\mathrm{Desc}(v)$ passes through v. We use the condition

$$\Delta_{c^{\mathrm{gt}}}(e) \;\geq\; \Delta_v(e) \qquad \text{for all } e \in \mathrm{Desc}(v). \tag{52}$$

This is the single behavioral condition of this appendix, and it has a direct reading: on the part of the branch that lies strictly below v, replacing the corrupted transition at its origin removes at least as much violation as replacing it at v, because restoring v leaves the corrupted context that feeds the path from c^{gt} to v in place. The condition is not imposed at v itself, where $\Delta_v(v) = V(v)$ holds by construction and restoring the origin need not clear v completely. It is also checkable rather than hypothetical: on labeled trajectories in which both c^{gt} and v are among the restored candidates, both sides are quantities computed during attribution.

Property 5 (responsibility margin). Write $S = \mathrm{Desc}(v)$ for the shared sub-branch, $X = (\{c^{\mathrm{gt}}\} \cup \mathrm{Desc}(c^{\mathrm{gt}})) \setminus (\{v\} \cup S)$ for the exclusive segment (which contains c^{gt} itself with weight $\psi^0 = 1$ and own term $\Delta_{c^{\mathrm{gt}}}(c^{\mathrm{gt}}) = V(c^{\mathrm{gt}})$), and $\zeta = \psi^{\,d(c^{\mathrm{gt}},v)}$. Under Condition 1 and Property 4,

$$R_{\mathrm{cf}}(c^{\mathrm{gt}}) - R_{\mathrm{cf}}(v) \;\geq\; \underbrace{\sum_{e \in X} \psi^{\,d(c^{\mathrm{gt}},e)} \Delta_{c^{\mathrm{gt}}}(e)}_{\text{explained only by the origin}} \;-\; \underbrace{\big[V(v) - \zeta\, \Delta_{c^{\mathrm{gt}}}(v)\big]}_{\text{symptom's own term}} \;-\; (1-\zeta) \underbrace{\sum_{e \in S} \psi^{\,d(v,e)} \Delta_{c^{\mathrm{gt}}}(e)}_{\text{shared sub-branch}}. \tag{53}$$

Consequently $R(c^{\mathrm{gt}}) > R(v)$ whenever the right-hand side exceeds the penalty difference $\lambda_{\mathrm{pen}}\big(\mathcal{P}(c^{\mathrm{gt}}) - \mathcal{P}(v)\big)/\omega$.

Support. Split $R_{\mathrm{cf}}(c^{\mathrm{gt}})$ over X, $\{v\}$, and S. The term at v is $\zeta \Delta_{c^{\mathrm{gt}}}(v)$, and $R_{\mathrm{cf}}(v)$ contributes $\Delta_v(v) = V(v)$ at weight $\psi^0 = 1$. For $e \in S$ every path from c^{gt} passes through v, so $d(c^{\mathrm{gt}}, e) = d(c^{\mathrm{gt}}, v) + d(v, e)$ and $\psi^{\,d(c^{\mathrm{gt}},e)} = \zeta\, \psi^{\,d(v,e)}$; Condition 1 then gives $\zeta \Delta_{c^{\mathrm{gt}}}(e) - \Delta_v(e) \geq -(1 - \zeta)\Delta_{c^{\mathrm{gt}}}(e)$ on S. Subtracting $R_{\mathrm{cf}}(v)$ and collecting the three parts gives Eq. (53). □

Property 5 is the analytical counterpart of the design decisions in Sec. A.6.3, and it makes explicit both what the rule gains over magnitude ranking and where it can still fail. Summing over the candidate itself as well as its descendants is what puts $V(c^{\mathrm{gt}})$ into the exclusive segment at full weight. The symptom's own violation does enter the margin: since $V^{\mathrm{res}}_{c^{\mathrm{gt}}}(v) \geq 0$ we have $\Delta_{c^{\mathrm{gt}}}(v) \leq V(v)$, so the middle term is at least $(1 - \zeta)V(v)$. What changes relative to magnitude ranking is its weight. Magnitude ranking selects c^{gt} only if $V(c^{\mathrm{gt}}) > V(v)$; when restoring the origin clears the symptom ($\Delta_{c^{\mathrm{gt}}}(v) \approx V(v)$), responsibility charges $V(v)$ only at weight $1 - \zeta$ (0.5 for a one-hop symptom at the default ψ=0.5) and credits the origin with the rest of the exclusive segment. The rule therefore tolerates a bounded amount of amplification, and this covers part of the regime of Sec. D.3 in which magnitude ranking already fails. It does not tolerate arbitrary amplification: if $V(v)$ or the mass on the shared sub-branch is large relative to the exclusive segment, or if restoring the origin leaves much of v's violation in place, the right-hand side can be negative and the symptom can outrank the source. This is one source of the non-zero mean hop distances in Table 2. The Hit@1 and mean-hop results in Table 2 and the comparison with TRACE-MAS (w/o CVA) in Fig. 2 are the empirical test of this mechanism.

D.5 STABILITY OF REPAIR SELECTION AND SIZE OF THE ROLLOUT ERROR

Repairs are ranked by the rollout estimate of Sec. 5.4 rather than by executing every candidate, so two questions arise: how selection behaves under rollout error, and how large that error is for the rollout we actually use. Write $\hat{\mathcal{J}}(u)$ for the rollout estimate of the repair objective and $\mathcal{J}(u)$ for its realized value after committing u and re-executing $\mathrm{Desc}(c^\star)$, both defined in Appendix A.7.2, and $u^\star = \arg\min_{u \in \mathcal{U}(c^\star)} \hat{\mathcal{J}}(u)$, $u^{\mathrm{opt}} = \arg\min_{u \in \mathcal{U}(c^\star)} \mathcal{J}(u)$.

Property 6 (selection is stable up to twice the offset-free error). Suppose there is a constant $\bar{b}$, which may depend on the trajectory but not on u, such that

$$\sup_{u \in \mathcal{U}(c^\star)} \left| \hat{\mathcal{J}}(u) - \mathcal{J}(u) - \bar{b} \right| \;\le\; \epsilon'. \tag{54}$$

Then $\mathcal{J}(u^\star) \le \mathcal{J}(u^{\mathrm{opt}}) + 2\epsilon'$.

Support. Selection is unchanged if $\bar{b}$ is subtracted from every $\hat{\mathcal{J}}(u)$, so $\mathcal{J}(u^\star) \le \hat{\mathcal{J}}(u^\star) - \bar{b} + \epsilon' \le \hat{\mathcal{J}}(u^{\mathrm{opt}}) - \bar{b} + \epsilon' \le \mathcal{J}(u^{\mathrm{opt}}) + 2\epsilon'$. □

Corollary 1 (exact selection under a rollout margin). Under Eq. (54), if $\hat{\mathcal{J}}(u) - \hat{\mathcal{J}}(u^\star) > 2\epsilon'$ for every $u \neq u^\star$, then $u^\star = u^{\mathrm{opt}}$.

Support. For $u \neq u^\star$, $\mathcal{J}(u) \ge \hat{\mathcal{J}}(u) - \bar{b} - \epsilon' > \hat{\mathcal{J}}(u^\star) - \bar{b} + \epsilon' \ge \mathcal{J}(u^\star)$. □

Property 6 needs no assumption about which operator is correct, and the offset $\bar{b}$ makes it insensitive to any error shared by all operators on a trajectory; $\bar{b} = 0$ recovers the uniform form. Corollary 1 turns the premise into a certificate that can be checked per trajectory once ϵ' is known. What remains is to show that ϵ' is small for the rollout of Appendix A.7.2, which the next property does by tracing the error to the two places where rollout and re-execution differ.

For each u, let $\mathcal{A}(u)$ be the (event, channel) pairs that the rollout sets to the contract mean (the edited channel of $c^\star$ and all channels of the regenerated window events), and $\mathcal{B}(u)$ the remaining pairs in the window, i.e. the two unedited channels of $c^\star$ and all channels of the carry-over events. For a pair (i, ch), let $\hat{V}_{i,\mathrm{ch}}$ and $V^{\mathrm{re}}_{i,\mathrm{ch}}$ be its weighted channel term (V_h, βV_m, or γV_s) in the rollout and in the re-execution, and define

$$\nu(u) = \sum_{(i,\mathrm{ch}) \in \mathcal{B}(u)} \left| \hat{V}_{i,\mathrm{ch}} - V^{\mathrm{re}}_{i,\mathrm{ch}} \right|, \qquad \theta(u) = \left| \hat{\mathcal{L}}_{\mathrm{task}}(u) - \mathcal{L}_{\mathrm{task}}(u) \right| \in [0, 1]. \tag{55}$$

Property 7 (rollout error of TRACE-MAS). Suppose re-execution under u produces the same K window events as the rollout (matched event by event), which holds for any operator that does not add or remove events among the first K descendants of $c^\star$. (a) For every such u,

$$\left| \hat{\mathcal{J}}(u) - \mathcal{J}(u) \right| \;\le\; \sum_{(i,\mathrm{ch}) \in \mathcal{A}(u)} V^{\mathrm{re}}_{i,\mathrm{ch}} \;+\; \nu(u) \;+\; \lambda_1\, \theta(u). \tag{56}$$

(b) Suppose that each re-executed regenerated window event is a benign event of its context, and that the re-executed edited channel of $c^\star$ is distributed as that channel of a benign event of its context, in both cases exchangeable with the benign calibration events of Eq. (30). Let τ be the threshold on the scale of V (Sec. D.1) with $\alpha \ge 1$. Then, with probability at least $1 - |\mathcal{U}(c^\star)|\,(K{+}1)\big(1 - \delta + \frac{1}{n+1}\big)$,

$$\sup_{u \in \mathcal{U}(c^\star)} \left| \hat{\mathcal{J}}(u) - \mathcal{J}(u) \right| \;\le\; (K{+}1)\,\tau \;+\; \max_u \nu(u) \;+\; \lambda_1 \max_u \theta(u), \tag{57}$$

so Eq. (54) holds with $\bar{b} = 0$ and ϵ' equal to the right-hand side.

Support. (a) The cost $\kappa(u)$ is the same deterministic quantity in both, so it cancels. The rollout scores every pair in $\mathcal{A}(u)$ as zero (Eq. (38)), whereas re-execution scores it as $V^{\mathrm{re}}_{i,\mathrm{ch}} \ge 0$; pairs in $\mathcal{B}(u)$ contribute at most $\nu(u)$; and the task terms differ by $\lambda_1 \theta(u)$. The triangle inequality gives Eq. (56). (b) For each u, the first term of Eq. (56) is a sum of at most $K{+}1$ quantities: the full score of each of the at most K regenerated window events, and the edited channel term of $c^\star$, which is at most the full score of a benign event with that channel. By Property 2 and $\alpha \ge 1$, each exceeds τ with probability at most $1 - \delta + 1/(n+1)$; a union bound over the at most $|\mathcal{U}(c^\star)|(K{+}1)$ quantities bounds the first term by $(K{+}1)\tau$ for every u simultaneously. □

Property 7 ties the error to the design choices of Appendix A.7.2 and to the operating point of Sec. A.9. First, the only term that grows with the depth is $(K{+}1)\tau$, and the failure probability grows with K as well, so deeper rollouts can only loosen the guarantee; at the deployed $K{=}1$ the first term is 2τ, and with $\delta{=}0.99$ and $\alpha{=}1$ the bound holds with probability at least $1 - 2|\mathcal{U}(c^\star)|\,(0.01 + 1/(n+1))$, e.g. $1 - 6\,(0.01 + 1/(n+1))$ with one candidate per operator type. Deeper rollouts

thus buy no tighter guarantee, which is consistent with TCR being unchanged from $K{=}1$ to $K{=}6$ in Fig. 6(c). Second, the first term is on the benign scale: τ is the level below which a benign event stays with probability at least $\delta - 1/(n+1)$, whereas an operator that leaves a corrupted field in place keeps that field's carry-over violation in $\hat{\mathcal{J}}$, whose expected size is, by the channel-wise form of Property 3, the benign mean of that channel plus $\|b\|^2_{\tilde{\Sigma}^{-1}}$, with b the shift the field induces relative to the repaired context. By Corollary 1, rollout selection is therefore exact whenever the rollout objective of every other operator, which contains the carry-over violation that operator leaves in place, exceeds that of $u^\star$ by more than $2\epsilon'$. Third, $\theta(u)$ is zero whenever re-execution invokes the template steps the rollout predicts, and $\nu(u)$ is zero whenever the carry-over events reproduce their logged transitions under the same contexts as in the rollout; neither is controlled analytically, which is why Appendix C.3 measures both on the paired pool $\mathcal{P}$. There, the matched-window precondition holds on every trajectory and rollout selection coincides with exhaustive re-execution, so whatever value gap ν and θ leave is smaller than the separation between operators, which is what Corollary 1 needs; the experiment bounds the gap through its effect on the ordering rather than measuring it directly. The assumption in (b) is the one the rollout itself makes, namely that an event whose inputs are repaired behaves benignly; it can fail for an operator that does not remove the corruption, in which case $\mathcal{J}(u)$ exceeds the bound for that u. The cost term $\lambda_2\kappa(u)$ biases selection in a definite direction: between operators with similar violation and task terms, the rule prefers the smaller intervention, which is consistent with the low operator cost reported in Table 9.

D.6 Cost Accounting

The pipeline of Sec. A.8 has a cost that follows from the definitions above rather than from measurement. Detection evaluates the contract once per event, at $O(|\mathcal{V}|\, d_z^2)$ time after context encoding. Attribution restores the top C candidates and rescores their descendants, at $O\big(C \cdot \max_c |\mathrm{Desc}(c)| \cdot d_z^2\big)$; without gating on Ω and on C, scoring every ancestor of every violated event would scale with $|\mathcal{V}| \cdot \max_c |\mathrm{Desc}(c)|$, which is the comparison the C sweep in Fig. 6 makes concrete. Repair selection adds $O\big(|\mathcal{U}(c^\star)| \cdot (K{+}1) \cdot d_z^2\big)$ for the candidate operators at $c^\star$ (the edited event and K window events per operator, Appendix A.7.2). The re-executions of every candidate in Appendix C.3 are an offline validation and are not part of this cost.

Two structural points follow. Counterfactual scoring and rollout are executed in contract space, so the number of LLM calls added by TRACE-MAS is the number of *committed* repairs, not the number of candidates considered; and because C, K, and $|\mathcal{U}(c^\star)|$ are small constants, the added non-LLM work is linear in trajectory length. This is why the wall-clock profile in Fig. 6 is reported excluding bulk generation: the design moves the search off the generation path by construction.

D.7 Scope

The statements above support the components of Sec. 5 and are deliberately narrow. Properties 1–3 characterize what the detection score measures and what its benign calibration guarantees; they do not claim that every corruption produces a large margin, and Property 3 makes the dependence on the induced shift explicit instead of assuming separability. Sec. D.3 derives why magnitude ranking is not localization but does not predict the amount of amplification for a given workflow. Properties 4–5 compare the true source against symptoms on its own branch under one checkable condition, and they apply only when c^{gt} enters the candidate set, i.e. lies in Ω or within H hops above it and survives the top-C pre-ranking; they do not establish identifiability against an adversary that corrupts several events, nor optimality over the whole candidate set when Condition 1 fails, and Eq. (53) can be negative under strong amplification at a symptom. Property 6 and Corollary 1 bound the suboptimality of repair selection given rollout error, and Property 7 bounds that error for our rollout only under a matched window and the benign-regeneration assumption of Property 7(b), with the carry-over and template terms left to measurement (Appendix C.3); none of them covers sequences of several committed edits or the completeness of the operator space $\mathcal{U}(c^\star)$. The quantitative behavior of all these mechanisms is established empirically in Sec. 6 and Appendix C.