MAADBENCH: THE REFRESHABLE PARADIGM FOR ANOMALY DETECTION IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent studies report LLM-based multi-agent systems (MAS) fail at rates of 41%–87%, yet to our knowledge, no benchmark to date supports systematic anomaly detection (AD) for them. Building MAS AD benchmarks is hard because they must remain valid as LLM systems evolve: tasks may leak into training data and thus be memorized by LLMs, traces and anomaly patterns expire as backbones evolve, and labels must be provided reliably for each refresh. To address these challenges, we present **MAADBENCH** (**MA**: multi-agent; **AD**: anomaly detection), the first refreshable MAS AD benchmark designed for diverse evolving LLM backbones underlying the agents. MAADBENCH combines (1) sampled-and-coupled generative tasks over a $\approx 10^{37}$-task space to mitigate task leakage, (2) refreshable trace generation under configurable LLM backbones, and (3) automated provision of cost-free, deterministic step-level labels for fine-grained AD evaluation. Beyond offering the paradigm itself, we run MAADBENCH with five SOTA LLM backbones, and release the dataset **MAADBENCH-FULL** with 5,200 step-labeled traces. Benchmarking 25 AD methods on MAADBENCH dataset reveals substantial limitations in current approaches: they rely heavily on supervision, struggle with subtle MAS-specific anomalies, and lack robustness across LLM backbones. These gaps point to a rich research agenda for MAS-specific anomaly detection, with MAADBENCH providing a systematic and refreshable testbed for method development and evaluation. We open-source MAADBENCH-FULL at `https://huggingface.co/datasets/hww123/MAADBench-full`.

1 INTRODUCTION

LLM-based multi-agent systems (MAS) have emerged as a promising solution for complex collaborative tasks (Liang et al., 2024; Zhuge et al., 2023; Google DeepMind, 2026), but their reliability remains a major concern. Recent studies report failure rates of 41%–87% even in state-of-the-art frameworks (Cemri et al., 2025). Prior works study MAS reliability through failure analysis (Cemri et al., 2025; Deshpande et al., 2025; Zhang et al., 2025a; 2026) and robustness mechanisms (Wang et al., 2025a; Miao et al., 2025; Yang et al., 2024), both naturally connected to anomaly detection (AD) by identifying deviations from expected MAS execution. While AD has been extensively studied in traditional settings (Chandola et al., 2009; Pang et al., 2021; Han et al., 2022), it remains underexplored for MAS. This motivates us to evaluate established AD techniques for detecting MAS errors and identify gaps requiring MAS-tailored detectors.

MAS anomaly detection remains underexplored, in part because MAS execution and data traces differ fundamentally from traditional AD targets. MAS traces are textual outputs from evolving, multi-step execution processes (Wu et al., 2024; LangChainAI, 2024), rather than fixed generate-once instances such as images or tabular records. They correspond to open-vocabulary natural language outputs, where anomalies tend to be semantic, logic-rich, arising from reasoning and inter-agent communication errors (Cemri et al., 2025; Wang et al., 2025a), rather than numerical or categorical deviations as common in system telemetry.

This introduces new design challenges unique to MAS AD benchmarks. As illustrated in Figure 1, a MAS benchmarking pipeline produces tasks and traces, and ideally also the corresponding labels as benchmark artifacts. Static MAS benchmark artifacts, however, can rapidly expire as the underlying

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.

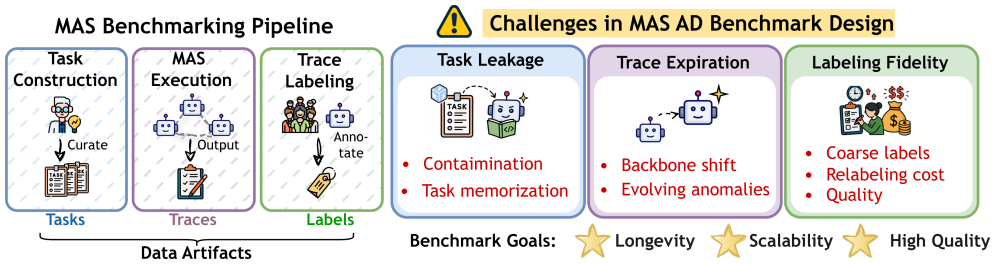


Figure 1: Overview of the MAS AD benchmarking pipeline (left) and the key design challenges (right) that must be addressed to achieve a long-lasting, scalable, and credible MAS AD benchmark.

LLMs plugged in for the agents continue to evolve at an unprecedented speed. To ensure the benchmark's longevity, scalability and high quality, we must address the following three challenges:

Challenge 1: Task Leakage. Benchmark leakage may cause LLMs to memorize answers rather than reason genuinely (Magar & Schwartz, 2022; Golchin & Surdeanu, 2023; Deng et al., 2024a;b; Xu et al., 2024b). For a MAS AD benchmark, this is especially harmful, as it may fail to capture authentic anomaly patterns that constitute the benchmark's core artifact.

Challenge 2: Trace Expiration. MAS traces and anomalies reflect the specific LLM backbone and version utilized to generate them (Liu et al., 2024; Chen et al., 2024). As LLMs evolve, their tackling of tasks may yield different traces and anomaly patterns, causing the prior collected traces to expire.

Challenge 3: Label Fidelity. Useful MAS labels must be fine-grained, reproducible, and high-quality, but achieving all three is difficult. Fine-grained step-level labels provide stronger supervision (Lightman et al., 2023; Wang et al., 2024) yet require costly inspection of long multi-agent traces. Worse, as traces are refreshed under evolving backbones, labels must also be regenerated on the fly. Balancing efficient label refresh with labeling cost and quality remains a key challenge.

Table 1: Comparison of SOTA MAS trace-analysis benchmarks across three core challenges. MAADBENCH addresses all three to support systematic MAS AD development and evaluation.

Benchmark	CH1: Task Leakage		CH2: Trace	CH3: Labeling Fidelity			AD
	Generative	Space †	Expiration	Step-level	Relabeling Cost §	Quality Guarantee	Support
MAST (Cemri et al., 2025)	× (fixed)	1,642	× (non-refresh.)	× (trace)	$$$$	× (human/LLM)	Limited
TRAIL (Deshpande et al., 2025)	× (fixed)	148	× (non-refresh.)	× (trace)	$$$$$$	× (human/LLM)	Limited
Silent Failures* (Pathak et al., 2025)	× (fixed)	5,169	× (non-refresh.)	× (trace)	$$$	× (human)	Limited
Who&When (Zhang et al., 2025a)	× (fixed)	184	× (non-refresh.)	✓ (step)	$$$$$	× (human/LLM)	Limited
AgenTracer (Zhang et al., 2026)	× (fixed)	2,500+	× (non-refresh.)	✓ (step)	$$	× (LLM-assisted)	Limited
MAADBENCH (ours)	**✓ (generative)**	$\approx 10^{37}$	**✓ (refresh.)**	**✓ (step)**	**$**	**✓ (deterministic)**	**Full**

*Dataset not yet publicly released. § Relabeling cost per trace (Appendix C). † Task space: prior benchmarks limited to released static traces, while our MAADBENCH expands space through generation (Appendix D.4).

State-of-the-Art Benchmarks on MAS Trace Analysis. We characterize existing MAS trace-analysis benchmarks in Table 1. Recent works have built MAS trace benchmarks by collecting and annotating traces from deployed systems (Cemri et al., 2025; Deshpande et al., 2025; Zhang et al., 2025a; 2026; Pathak et al., 2025). These efforts include *MAST* (Cemri et al., 2025) for failure taxonomies, *TRAIL* (Deshpande et al., 2025) for error localization, *Who&When* (Zhang et al., 2025a) for agent/step attribution, and *AgenTracer* (Zhang et al., 2026) for automated annotation via replay or fault injection. *Silent Failures* (Pathak et al., 2025) is closest in spirit to MAS AD, providing two labeled datasets for anomaly detection, but neither has been publicly released. While valuable, these benchmarks remain tied to fixed task sets (*CH 1*), static trace collections (*CH 2*), and one-time human or LLM labeling tied to specific backbones (*CH 3*), limiting their use as evolving infrastructures for MAS AD development and evaluation. A comprehensive survey of related works is in Appendix A.

MAADBENCH: The Refreshable Benchmarking Paradigm for MAS AD. MAADBENCH addresses the three core challenges of MAS AD benchmarking. To address *task leakage*, it samples and couples hidden-dependency atomic tasks at configurable complexity levels, yielding a huge combinatorial task space rather than a fixed set. To handle *trace expiration*, it enables trace refreshability by executing the tasks under new LLM backbones. To ensure *label fidelity*, each subtask is generated with oracle ground truth, enabling automatic deterministic step-level labeling without requiring human or LLM annotation. Further, MAADBENCH elicits *diverse MAS capabilities* across domains, including math,

coding, planning, reasoning, and communication, with multiple task difficulties, producing rich anomaly types for comprehensive AD evaluation.

Our contributions are summarized as follows:

1. **MAADBENCH: The Refreshable Benchmarking Paradigm for MAS AD.** We establish principles for refreshable MAS AD benchmarking, realize them through bottom-up task construction, and instantiate them in MAADBENCH to address task leakage, trace expiration, and label fidelity.
2. **Multi-LLM-as-Backbone MAS AD Datasets.** We run MAADBENCH with five LLM backbones and multiple configurations to generate and release MAADBENCH-FULL, containing 5,200 MAS traces with deterministic step-level labels.
3. **Broad Evaluation Across AD Methods and LLM Backbones.** We profile MAS anomalies across action types, LLM backbones, observability, and propagation. We evaluate 25 AD methods across diverse supervision levels and detector families, spanning tabular, graph-based, MAS-specific, and LLM-as-a-judge approaches. This evaluation studies their performance, robustness across anomaly profiles, and propagation capture.
4. **New Findings on MAS Anomaly Detection.** Our evaluation shows that the MAS AD problem is hard: SOTA AD methods on MAS rely heavily on supervision, with a $\sim 1.5\times$ larger gap than tabular AD (Han et al., 2022). Their performance varies across LLM backbones, while the long-horizon reasoning, task-failing, and propagation-origin anomalies remain hardest to detect. These findings outline a research agenda for MAS-tailored AD that prioritizes less supervision, backbone robustness, semantic understanding, and root-cause awareness.

2 MAADBENCH CONSTRUCTION AND INSTANTIATION

We now present the design principles and realization of MAADBENCH. A MAS AD benchmark should satisfy two goals: (1) complex tasks that elicit multi-agent coordination and diverse anomalies, and (2) automated, fine-grained ground-truth labeling of anomalous agent behaviors. Existing benchmarks (Cemri et al., 2025; Deshpande et al., 2025; Zhang et al., 2026; 2025a) use complex collaborative tasks but leave task decomposition and execution to the MAS and annotate traces afterward. This separation of task construction, execution, and labeling makes automated generation of reliable step-level labels difficult.

Bottom-up Oracle-Guided Construction. To support diverse and complex tasks while retaining deterministic fine-grained labels, we construct the benchmark bottom-up from atomic tasks, all of which are equipped with verified ground truth. This bottom-up design allows atomic tasks from different domains and benchmarks to be flexibly plugged into the framework. Atomic tasks are composed through *oracle dependencies*, where the output of one task may serve as a prerequisite or input to another. Given task ground truth and oracle dependencies, labels can be deterministically propagated along intra-level input-output dependencies and across higher levels of the task hierarchy. *This principle enables large-scale combinatorial task generation with flexible configuration yet with deterministic labels at multiple task levels.*

An Instantiation of MAADBENCH. The atomic task space of MAADBENCH spans diverse capabilities, including mathematical and code reasoning, planning, distraction handling, environment interaction, tool use, and verification. They can be sampled from current benchmarks with deterministic ground truth (e.g., LiveCodeBench-Execution (Jain et al., 2025), GSM-Hard (Gao et al., 2023), GSM-Symbolic (Mirzadeh et al., 2025), *e.t.c*), extended to future benchmarks as LLM capabilities evolve, or generated from parameterized templates with algorithmic ground truth. To compose these atomic tasks into a meaningful MAS scenario, we use an escape room as a *metaphor for task composition*: each agent *action* executes an atomic task; atomic tasks (*actions*) are composed into intermediate composite tasks (*puzzles*), which are further composed into top-level composite tasks (*rooms*). The design supports configurable task counts and dependency complexity across difficulty levels, yielding diverse task instances that mitigate memorization and task leakage. Figure 2 summarizes one concrete instantiation of MAADBENCH, including its task configuration, capabilities, and ground-truth provenance; full details and a room example are provided in Appendix D.

MAS Workflow and Trace Collection. Following the task design above, we build a multi-agent system where specialized agents execute the workflow in Figure 2 and communicate through structured JSON messages containing intermediate reasoning and outputs. We record the acting agent, action

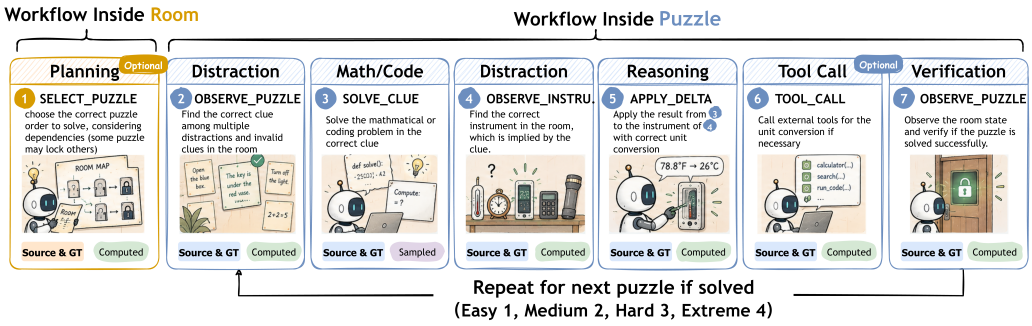


Figure 2: **MAADBench Instantiation and MAS Execution. 1) Workflow:** The execution workflow follows the same hierarchy: at the room level, the MAS resolves cross-puzzle dependencies; at the puzzle level, it executes atomic tasks with input-output dependencies. **2) Top-level Composite Task (Room):** The MAS first needs to reason about the correct puzzle execution order under cross-puzzle dependencies, as a planning task. **3) Intermediate Composite Task (Puzzle):** Each selected puzzle needs to be solved through a sequence of interdependent agent actions examining different capabilities. The SOLVE_CLUE tasks are sampled from GSM-Hard (Math) and LiveCodeBench-Execution (Code), while the remaining actions are generated from parameterized templates with algorithmic ground truth. **4) Overall Task Difficulty:** We define four room difficulty levels: Easy, Medium, Hard, and Extreme, by increasing the number of puzzles and the complexity of their cross-puzzle dependencies.

type, input/output messages, and rich runtime telemetry (e.g., token usage, latency, *e.t.c*). Additional implementation details on the MAS harness and tool calls are provided in Appendix E.

3 PROFILING MAS ANOMALIES: ACTIONS, BACKBONES, OBSERVABILITY, AND PROPAGATION

Building on the above design, we now deploy the MAADBENCH framework to generate a sample dataset, i.e., MAADBENCH-FULL, and use it to profile the emergent MAS anomalies in terms of LLM capabilities, LLM backbones, observability, and propagation patterns. *Note MAADBENCH provides automatic mapping from anomalies to the MAST (Cemri et al., 2025) failure taxonomy*, enabling benchmark labels to be interpreted through existing MAS failure categories; details in Appendix F.

MAS Execution Setup. Following the task instantiation in Figure 2, we utilize two core SOLVE_CLUE domains, *Math* and *Code*. For each domain, we generate 100 rooms, with 25 rooms at each difficulty level, resulting in 200 unique rooms. We execute MAS using five popular LLM backbones: Claude Sonnet 4.6 (Anthropic, 2025), Claude Opus 4.8 (Anthropic, 2026), DeepSeek-R1 (Guo et al., 2025), GPT-4.1 (OpenAI, 2025) and GPT-5.4 (OpenAI, 2026). For each backbone, we run both *Tool* and *No-Tool* modes, with external tool use enabled or disabled, respectively. LLMs without temperature control (e.g., Claude Opus 4.8) are run with the default settings, while the remaining LLMs run with temperatures of 0.0, 0.3, and 0.6. Each trace is automatically annotated with ground-truth labels at the room, puzzle, and action levels. Trace token costs are reported in Table 7 in the Appendix.

Table 2: Room-Level and Puzzle-Level MAS Success Rate (↑) Across LLM Types on MAADBENCH-FULL. **Best** and worst values are highlighted. All rooms are fixed across all LLM instantiations to ensure fair comparison. Success Rate (SR) is verified by the final Room/Puzzle output.

LLM	# Room (×Temp)	Room-Level SR					Puzzle-Level	
		Easy	Medium	Hard	Extreme	Overall	Attempted / Total	SR
Claude Opus 4.8	400 (×1)	**70.0%**	**63.0%**	**29.0%**	**30.0%**	**48.0%**	805 / 1000	**61.0%**
Claude Sonnet 4.6	400 (×3)	52.3%	35.3%	11.0%	19.3%	29.5%	2065 / 3000	41.4%
DeepSeek-R1	400 (×3)	66.7%	42.0%	12.3%	22.0%	35.8%	2054 / 3000	43.9%
GPT-4.1	400 (×3)	46.3%	19.0%	3.0%	0.0%	17.1%	1521 / 3000	21.0%
GPT-5.4	400 (×3)	60.0%	27.0%	10.3%	0.3%	24.4%	1689 / 3000	30.0%

MAADBENCH is Challenging for SOTA LLMs. We report the overall success rate of the MAS execution on the room and puzzle level in Table 2. At both levels, performance drops sharply as

the overall difficulty increases. Claude Opus 4.8 ranks first on both levels, while GPT-4.1 ranks last across both levels.

Table 3: Success Rate (↑) of Actions on MAADBENCH-FULL. For every action, we annotate the primary LLM capability it evaluates.

LLM	SEL._PUZZLE *(Planning)*	OBS._CLUE *(Distraction)*	SOL._CLUE *(Math)*	OBS._INSTRU *(Distraction)*	TOOL_CALL *(Tool Use)*	APL._DELTA *(Reasoning)*	OBS._PUZZLE *(Verification)*	Overall
Claude Opus 4.8	94.8%	**99.8%**	80.4%	**97.8%**	**73.1%**	**75.8%**	98.9%	**89.1%**
Claude Sonnet 4.6	97.1%	97.5%	**80.5%**	95.4%	64.9%	60.1%	**100.0%**	85.0%
DeepSeek-R1	**100.0%**	46.2%	63.0%	73.3%	60.8%	64.1%	97.0%	68.4%
GPT-4.1	70.9%	97.2%	71.3%	94.3%	63.2%	41.5%	98.4%	78.3%
GPT-5.4	65.6%	98.3%	78.7%	93.5%	67.0%	53.3%	99.9%	82.2%
Average	85.7%	87.8%	74.8%	90.9%	65.8%	59.0%	98.8%	80.6%

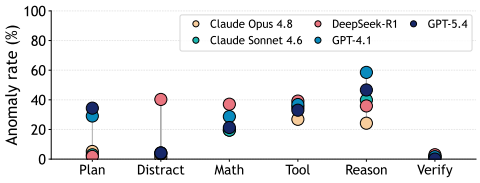


Figure 3: Anomaly rate by capability × LLMs.

Anomalies Across Capabilities and LLM Backbones. We characterize anomalies at the finest granularity, *i.e.*,, the action level. As shown in Table 3 and Figure 3, anomaly patterns are strongly *capability-specific* and vary substantially across LLM backbones. Verification action is consistently reliable, with anomaly rates below 3%, whereas tool use remains challenging across models ($\sim$ 27–39%). Reasoning shows even larger variation, ranging from roughly 24% to 59%, indicating substantial backbone-specific differences. Planning and distraction handling are similarly model-dependent, with anomaly rates varying from near 0% (zero) to around 40% across backbones. It is interesting to observe that each backbone exhibits a distinct anomaly pattern over different capabilities.

Observation 1 [Backbone Effect]. *Stronger LLMs do not uniformly reduce anomalies; instead, each backbone exhibits distinct capability and anomaly profiles.*

Silent vs. Loud Anomalies. We divide failures into *silent*, *i.e.*, failed actions within solved puzzles, and *loud* anomalies, *i.e.*, failed actions within failed puzzles. Surprisingly, Table 4 shows that overall, 7.7%–30.4% anomalies are silent. By action, planning errors (SELECT_PUZZLE) are always silent, the reason may be that they affect long-term planning rather than current puzzle outcomes; final-step errors (APPLY_DELTA) are always loud, as incorrect updates directly lead to puzzle failure; and intermediate actions (SOLVE_CLUE, OBSERVE_INSTRUMENT, TOOL_CALL) show mixed behavior. Both types matter for MAS monitoring: *loud anomalies are task-fatal, while silent anomalies reveal latent behavioral deviations.*

Observation 2 [Observability Gap]. *MAS anomalies exhibit different levels of observability, requiring AD methods to remain effective across both explicit task failures and subtler intermediate deviations.*

Table 4: Silent failure rate by model and failure type on MAADBENCH-FULL. Formatting errors (**Fmt Err**), *e.g.*, invalid JSON outputs, separated from action failures. Each column reports fraction of silent failures among all failures of corresponding type. "–" indicates no such failure occurred.

LLM	Overall	Fmt Err	SEL._PUZZLE	OBS._CLUE	SOL._CLUE	OBS._INSTRU	TOOL_CALL	APL._DELTA	OBS._PUZZLE
Claude Opus 4.8	9.0%	9.1%	100.0%	0.0%	10.2%	16.7%	14.5%	0.0%	0.0%
Claude Sonnet 4.6	7.7%	18.8%	100.0%	0.0%	7.9%	42.1%	10.5%	0.0%	–
DeepSeek-R1	30.4%	34.1%	–	–	1.5%	50.0%	7.5%	0.0%	100.0%
GPT-4.1	12.2%	10.9%	100.0%	13.9%	13.5%	10.8%	25.5%	0.0%	80.0%
GPT-5.4	11.4%	18.5%	100.0%	10.3%	4.2%	20.0%	2.5%	0.0%	0.0%

Anomaly Propagation. Figure 4 shows that anomalies generally propagate along the workflow order. Across GPT and Claude backbones, early perception failures (OBSERVE_CLUE) often lead to downstream failures in SOLVE_CLUE, TOOL_CALL, and especially APPLY_DELTA. This indicates that incorrect upstream observations can contaminate both reasoning and final execution. APPLY_DELTA acts as a common sink for propagated errors, since failures in solving, instrument observation, or tool use often surface as incorrect state updates. DeepSeek-R1 shows a less concentrated pattern, with failures spread more broadly across observation, solving, and tool-use steps. Overall, propagation is shaped both by the task workflow and by backbone-specific failure profiles.

Observation 3 [Heterogeneous Propagation]. *Anomaly propagation patterns are shaped by both the causal structure of the workflow and the LLM backbone.*

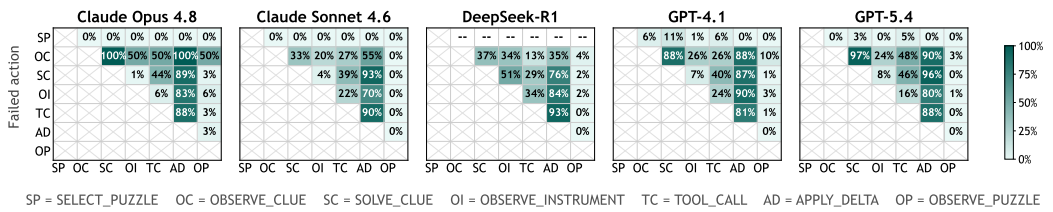


Figure 4: Action-level anomaly propagation by LLMs. **Row**: failed source action, **Column**: subsequent failed actions. **Colored cells**: propagation rates, as fraction of source failures followed by a downstream failure of target action. Darker colors indicate stronger propagation. **"–" cells**: no source failures. **Cross-hatched cells**: invalid action pairs.

4 Evaluating Anomaly Detection Strategies on MAADBench

We now use MAADBench traces and labels to evaluate AD methods. We select a core subset MAADBench-Eval from MAADBench-Full with temperature $= 0$, as they yield more stable model behavior for AD evaluation. Our evaluation assesses the effectiveness of AD methods on MAS anomalies and characterizes MAS anomaly detectability to inform future research into MAS-tailored detectors. Specifically, we conduct experiments to answer **RQ1**(§4.2): Which classes of AD methods are effective for MAS anomaly detection? **RQ2** (§4.3): Which MAS anomalies are most detectable by which methods? **RQ3** (§4.4): How early in the long trace can AD methods capture anomaly propagation? Below, each subsection addresses one of the RQs; including analysis of results, key takeaways, and proposal of future MAS AD research.

4.1 Experimental Setup

Dataset. MAADBench-Eval contains 17,180 labeled actions, with an anomaly ratio of 19.68%. We split the dataset into training and testing sets using an 8:2 stratified split. To support different AD families, we construct method-specific inputs, including tabular features such as token counts and embeddings, graph representations, MAS-aware interaction graphs, and raw-text traces for LLM-based detectors. Full data pre-processing details are provided in Appendix G.1.

Methods and Evaluation Metrics. We evaluate 25 AD methods across *four supervision settings:* unsupervised, one-class classification (OCC), semi-supervised with 5% labeled anomalies following existing benchmarks (Han et al., 2022), and fully supervised. These methods span four families: tabular, graph-based, MAS-specific, and LLM-as-detector methods. Representative baselines include XGBoost (Chen & Guestrin, 2016), Random Forest (Breiman, 2001), DevNet (Pang et al., 2019), DeepSAD (Ruff et al., 2019), OC-SVM (Schölkopf et al., 2001), DOMINANT (Ding et al., 2019), TAM (Qiao & Pang, 2023), GGAD (Qiao et al., 2024), G-Safeguard (Wang et al., 2025a), BlindGuard (Miao et al., 2025), and Gemma4 (Google DeepMind, 2026); with full list of methods reported in Figure 5, and the method details and computation resources in Appendix G. We report AUC-ROC here, with F1-score, accuracy, and balanced accuracy in the Appendix H.

4.2 Overall Effectiveness of Existing AD Methods

Detection Performance Across Supervision Levels. Figure 5 reports each method's mean AUCROC. Method performance generally increases as stronger anomaly supervision becomes available. LLM-based unsupervised detectors and the strongest graph-based OCC methods extract more meaningful signal. Supervision (Semi- and fully-supervised) yields substantial gains, with top methods (XGBoost, G-Safeguard, Random Forest) reaching AUROC ≥ 0.80. Notably, the supervision gap on MAADBench ($\sim$0.28 AUROC) is substantially larger than on tabular ADBench ($\sim$0.20 between best unsupervised and fully-supervised methods) (Han et al., 2022). This suggests that MAS AD is challenging for existing methods without explicit anomaly signals. However, large-scale step-level anomaly labels are often expensive to obtain in deployed MAS.

Takeaway 1: MAS Anomalies Are Particularly Difficult to Detect Without Supervision. Compared with traditional AD, MAS AD exhibits a substantially larger supervision gap, suggesting that MAS-specific anomaly signals are harder to discover without labels.

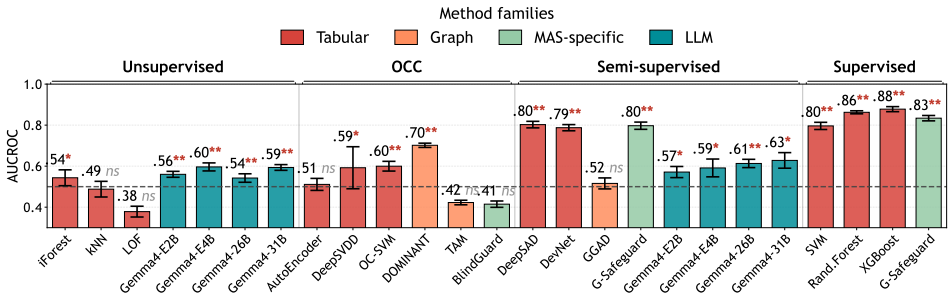


Figure 5: Overall AUCROC of AD methods categorized by supervision level and method family. LLM zero-shot methods are grouped as unsupervised, while LLM few-shot methods are grouped as semi-supervised. **Dashed line**: A random baseline with AUCROC=0.5 as reference. **Significance vs. Random**: ** $p < 0.01$, * $p < 0.05$, ns $p \geq 0.05$. More metrics with statistical tests are reported in Table 9 and 10 in Appendix H.

Detection Performance Across Method Families. We compare method families within each supervision level (Figure 5). The winning family shifts across supervision. As described before, only LLM zero-shot extracts meaningful signal (Gemma 0.54–0.60) in the unsupervised setting. However, performance shows no clear gain with model size. In contrast, few-shot supervision yields only modest overall improvement, but introduces a clearer scaling benefit, with larger Gemma variants performing better. This suggests that model scale becomes more useful when limited anomaly supervision is available. In OCC, graph-based DOMINANT performs best (0.70), while TAM remains near the random baseline (0.42), indicating that graph-based modeling is not uniformly beneficial for MAS traces. With semi- or full supervision, MAS-specific and tabular methods dominate: DeepSAD and G-Safeguard tie atop semi-supervised (0.80), while XGBoost leads supervised (0.88), followed by Random Forest (0.86) and G-Safeguard (0.83).

Takeaway 2: Designing for MAS Matters. The most effective methods exploit MAS-specific signals, *i.e.*, semantic understanding (LLMs with lower supervision) or MAS-specific agent interaction modeling.

Future Directions: Toward Low-cost and Label-efficient MAS AD. Our takeaways answer **RQ1**: existing AD methods are most effective for MAS AD only under strong supervision or are tailored to MAS-specific signals. Future MAS AD research should aim to close this gap with less-supervised and MAS-tailored methods. While LLMs offer strong semantic understanding of the abnormal traces, our cost analysis (Figure 10, Appendix G) shows a 2-order-of-magnitude cost gap between LLM-based and conventional detectors, making LLM-as-a-judge costly for long MAS traces. This motivates cost-effective LLM-based detection through model cascading (Nie et al., 2024), distillation (Hinton et al., 2015), or selective invocation (Ding et al., 2024). For non-LLM or hybrid directions, promising directions include better modeling of MAS semantics and agent interactions.

4.3 Detection Performance Across Different MAS Anomaly Types

We now study anomaly detectability across AD methods. We categorize MAS anomalies based on our profiling in §3, based on: (1) the action and LLM backbone in which they arise (Observation 1), and (2) their observability as silent or loud anomalies (Observation 2).

Anomaly Detectability vs. Action Success. Figure 6 reports each action's success rate and AUCROC across methods. Action success and detectability correlate only weakly: Solve_Clue achieves the highest average AUCROC (0.64) despite a moderate success rate (76.0%), while Observe_Puzzle has the highest success rate (99.5%) but slightly lower detectability (0.61). Notably, Select_Puzzle, which requires long-horizon reasoning over cross-puzzle dependencies, is the hardest to detect (avg AUCROC 0.51) despite a relatively high success rate (87.0%). This suggests that anomalies involving long-horizon reasoning may be particularly challenging for AD methods.

Takeaway 3: Existing AD Methods Struggle to Detect Anomalies in Long-Horizon Reasoning. Actions requiring long-horizon reasoning exhibit the lowest detectability, highlighting a key challenge for current AD methods.

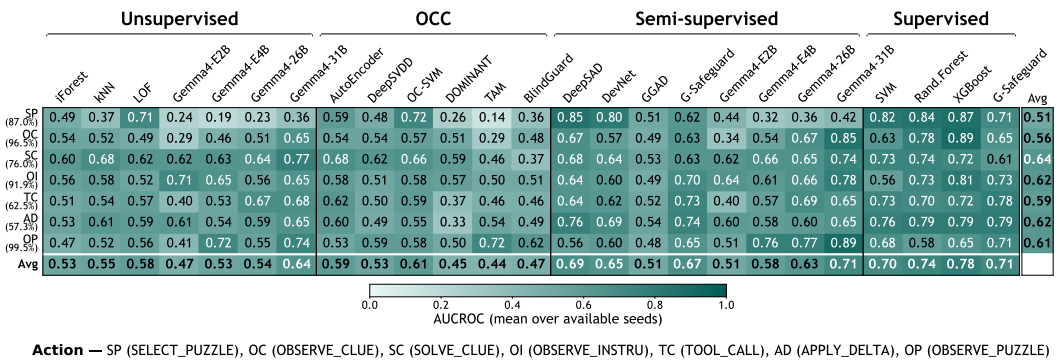


Figure 6: AUCROC of AD methods by actions (full results with std in Table 11 Appendix H). **Rows:** Six action types with success rate; **Bottom row:** avg AUCROC per method across actions. **Columns:** Method AUC. **Rightmost column:** avg AUCROC per action across methods.

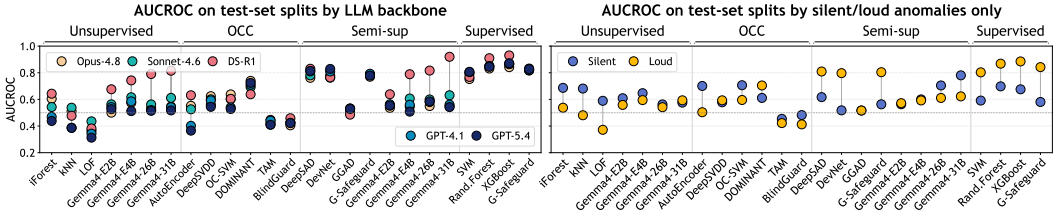


Figure 7: AD performance for different anomaly types. Evaluated **Left**: on test sets split by LLM backbone. **Right**: on test sets split by anomaly type (silent: puzzle solved; loud: puzzle failed).

Anomaly Detectability vs. LLM Backbones. Figure 7 (left) reports detection performance on test sets split by LLM backbone. Detection is generally easier on DeepSeek-R1 traces and harder on GPT-series traces. Backbone sensitivity is most pronounced for LLM-based detectors (Gemma variants), with AUCROC gaps approaching 0.38 (Gemma-4-31B few-shot) across backbones. In contrast, semi- and fully-supervised methods are substantially more stable across backbones, suggesting that anomaly supervision improves robustness to backbone-specific anomaly patterns.

Takeaway 4: Method Robustness Varies Across LLM Backbones. Effective evaluation of MAS AD therefore requires testing across multiple LLM backbones.

Anomaly Detectability vs. Observability. We compare the detectability of silent and loud anomalies, as defined in Table 4. Since silent format errors are easy to detect, we focus on the remaining harder silent anomalies. Figure 7(right) reveals a supervision-dependent observability gap. Under unsupervised and OCC settings, loud anomalies are often harder to detect than silent ones, indicating that task-fatal failures do not necessarily produce stronger local anomaly signals. With stronger non-LLM supervision, this trend reverses: semi- and fully-supervised detectors detect loud anomalies more reliably, with supervised methods reaching 0.80–0.88 AUCROC. In contrast, the performance of LLM detectors is much more robust to the observability of the anomalies.

Takeaway 5: Loud Anomalies Require Supervision. Loud anomalies can look locally plausible enough to evade detection; supervision primarily improves detection of these task-fatal failures.

Future Directions: Beyond Surface Anomaly Detection. Our takeaways answer **RQ2**: the hardest anomalies are logical, task-failing, and produced by well-formatted backbones. Although supervision and MAS-specific design help, these findings motivate three directions. First, logical reasoning failures call for richer signals beyond surface embeddings, including LLM-internal features such as hidden states (Zhou et al., 2026; Abdelnabi et al., 2025; Xu et al., 2026), attention patterns (He et al., 2025), and token-level uncertainty (Kossen et al., 2024; Zhang et al., 2025b). Second, task-failing anomalies may remain locally plausible, motivating consistency- or verification-based detectors such as inter-agent checks and debate (Li et al., 2026; Sun et al., 2025). Third, backbone-induced variance up to 0.30 AUROC motivates backbone-robust detectors and multi-backbone evaluation enabled by MAADBENCH (Yang et al., 2023).

4.4 Detection Performance under MAS Anomaly Propagation

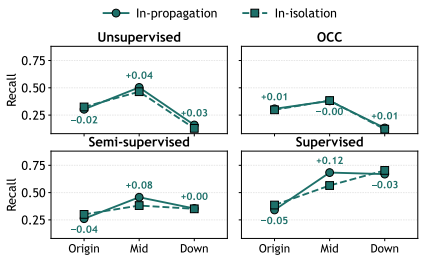


Figure 8: Recall on anomaly propagation chains. **In-propagation (solid)**: recall on an action inside propagation chains; **In-isolation (dashed)**: global recall on the same actions as reference. The delta of the two is annotated. $-$ means the anomalies are *harder* to detect inside a propagation chain, $+$ otherwise.

To analyze detection under anomaly propagation, we identify propagation chains as puzzles with multiple failed actions. We label each failed action by its chain role: **Origin** (first), **Mid**dle (intermediate), or **Down**stream (last). We report the average recall of each supervision family across these roles in Figure 8, since identifying propagation early is the focus.

Across supervision families, propagation has a role-dependent effect on anomaly detectability. Origin steps generally stay harder in propagation, with lower recall (delta down to -0.04) on origins than on isolated failures. The strongest propagation effect appears at middle steps, where recall increases for most families and reaches the largest gain under supervised methods ($+0.12$). Downstream effects are more mixed: unsupervised, OCC, and semi-supervised detectors show small gains, whereas supervised detectors perform slightly worse than their isolated reference. These results suggest that propagation most clearly exposes intermediate accumulated errors. A detailed per-method, per-role breakdown is provided in Figure 11 of Appendix H.

Takeaway 6: Root Causes Remain Hard to Detect. Propagation more consistently exposes accumulated middle-step anomalies, while root-cause failures often remain subtle and detector-dependent.

Future Directions: Detecting Subtle Root-Cause Anomalies. Our takeaways answer **RQ3**: in propagation chains, AD methods detect downstream anomalies more easily than in isolation but miss origin failures more often; stronger supervision widens this asymmetry. Two directions follow. First, AD should move beyond isolated step features to propagation-aware models that exploit inter-step dependencies, surfacing anomalies whose signals span multiple steps. Insights from causality-aware AD in related domains (Carvalho et al., 2023; Kim et al., 2025) can inform such designs. Related insights from process-reward modeling show that step-level progress signals improve credit assignment over outcome-only supervision (Setlur et al., 2025), motivating similar step-attribution mechanisms for MAS AD. Second, *root-cause localization* goes beyond detection by identifying which step initiates a cascade. Existing failure-attribution works for LLM agents (Wang, 2026; Zhang et al., 2025a; 2026) address this post-hoc on completed traces, while real-time MAS AD requires proactive localization during execution.

5 Conclusion

MAS anomaly detection needs a refreshable benchmarking paradigm to remain useful as LLMs evolve. We propose the innovative MAADBench paradigm that tackles the key challenges, instantiate MAADBench-Full as a labeled-trace dataset, and evaluate diverse SOTA AD methods on it. Results show MAS AD remains largely unsolved, motivating the need for the development of AD detectors tailored to multi-agent settings. As a key contribution, our MAADBench design principles, namely, generative tasks, reproducible traces, and deterministic step-level labels, future-proof MAS AD benchmarking and method development amid rapid LLM evolution.

Limitations. Although MAADBench covers diverse capabilities and domains, we acknowledge it does not fully exhaust the complete MAS design space, where variations in agent organization, communication, and memory strategies may introduce new anomaly surfaces and propagation patterns. A full discussion can be found in Appendix B.

Broader Impact. MAADBench provides an adaptive testbed for developing and evaluating future MAS anomaly detection methods. By enabling systematic MAS AD research on rapidly evolving LLMs and agent systems, MAADBench contributes to the broader goal of building reliable LLM-based multi-agent systems and thus furthers their future application for solving real-world problems.

AI USE DISCLOSURE

Generative AI tools were used to assist with coding, language editing, improving the readability of the manuscript, and checking formatting and presentation. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the final work. The research ideas, benchmark design, methodology, experiments, analyses, and conclusions are the authors' original contributions.

REFERENCES

Sahar Abdelnabi, Aideen Fay, Giovanni Cherubin, Ahmed Salem, Mario Fritz, and Andrew Paverd. Get my drift? catching llm task drift with activation deltas. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 43–67. IEEE, 2025.

Anthropic. Introducing claude opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, May 2026. Accessed: 2026-09-18.

AI Anthropic. System card: Claude opus 4 & claude sonnet 4. *Claude-4 Model Card*, 2025.

Liron Bergman and Yedid Hoshen. Classification-based anomaly detection for general data. In *International Conference on Learning Representations (ICLR)*, 2020.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104, 2000.

João Carvalho, Mengtao Zhang, Robin Geyer, Carlos Cotrini, and Joachim M Buhmann. Invariant anomaly detection under distribution shifts: A causal perspective. *Advances in Neural Information Processing Systems*, 36:56310–56337, 2023.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=fAjbYBmonr.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

Lingjiao Chen, Matei Zaharia, and James Zou. How is chatgpt's behavior changing over time? *Harvard Data Science Review*, 6(2), 2024.

Simin Chen, Yiming Chen, Zexin Li, Yifan Jiang, Zhongwei Wan, Yixin He, Dezhi Ran, Tianle Gu, Haizhou Li, Tao Xie, et al. Benchmarking large language models under data contamination: A survey from static to dynamic evaluation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10091–10109, 2025.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Chunyuan Deng, Yilun Zhao, Yuzhao Heng, Yitong Li, Jiannan Cao, Xiangru Tang, and Arman Cohan. Unveiling the spectrum of data contamination in language model: A survey from detection to remediation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16078–16092, 2024a.

Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. Investigating data contamination in modern benchmarks for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8706–8719, 2024b.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025.

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

Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM international conference on data mining*, pp. 594–602. SIAM, 2019.

Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Shahriar Golchin and Mihai Surdeanu. Time travel in llms: Tracing data contamination in large language models. *arXiv preprint arXiv:2308.08493*, 2023.

Google DeepMind. Gemma 4: Byte for byte, the most capable open models, April 2026. URL https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/. Blog post.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. Adbench: Anomaly detection benchmark. *Advances in neural information processing systems*, 35:32142–32159, 2022.

Pengfei He, Zhenwei Dai, Xianfeng Tang, Yue Xing, Hui Liu, Jingying Zeng, Qiankun Peng, Shrivats Agrawal, Samarth Varshney, Suhang Wang, et al. Attention knows whom to trust: attention-based trust management for llm multi-agent systems. *arXiv preprint arXiv:2506.02546*, 2025.

Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

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

HyunGi Kim, Jisoo Mok, Dongjun Lee, Jaihyun Lew, Sungjae Kim, and Sungroh Yoon. Causality-aware contrastive learning for robust multivariate time-series anomaly detection. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=EGpueKe6TP.

Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in llms. *arXiv preprint arXiv:2406.15927*, 2024.

LangChainAI. LangGraph. https://github.com/langchain-ai/langgraph, 2024. Accessed: 2024.

Zhuo Li, Yupeng Zhang, Pengyu Cheng, Jiajun Song, Mengyu Zhou, Hao Li, Shujie Hu, Yu Qin, Erchao Zhao, Xiaoxi Jiang, et al. March: Multi-agent reinforced self-check for llm hallucination. *arXiv preprint arXiv:2603.24579*, 2026.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 17889–17904, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pp. 413–422. IEEE, 2008.

Qinghua Liu and John Paparrizos. The elephant in the room: Towards a reliable time-series anomaly detection benchmark. *Advances in Neural Information Processing Systems*, 37:108231–108261, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Inbal Magar and Roy Schwartz. Data contamination: From memorization to exploitation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 157–165, 2022.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

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. *arXiv preprint arXiv:2508.08127*, 2025.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6129–6139, 2025.

Lunyiu Nie, Zhimin Ding, Erdong Hu, Christopher Jermaine, and Swarat Chaudhuri. Online cascade learning for efficient inference over streams. *arXiv preprint arXiv:2402.04513*, 2024.

OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025. Accessed: 2026-05-06.

OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026. Accessed: 2026-05-06.

Guansong Pang, Chunhua Shen, and Anton Van Den Hengel. Deep anomaly detection with deviation networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 353–362, 2019.

Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38, 2021.

Divya Pathak, Harshit Kumar, Anuska Roy, Felix George, Mudit Verma, and Pratibha Moogi. Detecting silent failures in multi-agentic ai trajectories. *arXiv preprint arXiv:2511.04032*, 2025.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Hezhe Qiao and Guansong Pang. Truncated affinity maximization: One-class homophily modeling for graph anomaly detection. *Advances in Neural Information Processing Systems*, 36:49490–49512, 2023.

Hezhe Qiao, Qingsong Wen, Xiaoli Li, Ee-Peng Lim, and Guansong Pang. Generative semi-supervised graph anomaly detection. *Advances in neural information processing systems*, 37: 4660–4688, 2024.

Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 427–438, 2000.

Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pp. 4393–4402. PMLR, 2018.

Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*, 2019.

Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pp. 4–11, 2014.

Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

Tom Shenkar and Lior Wolf. Anomaly detection for tabular data with internal contrastive learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=_hszZbt46bT.

Xiaoxi Sun, Jinpeng Li, Yan Zhong, Dongyan Zhao, and Rui Yan. Towards detecting llms hallucination via markov chain-based multi-agent debate framework. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2025.

Jianheng Tang, Fengrui Hua, Ziqi Gao, Peilin Zhao, and Jia Li. Gadbench: Revisiting and benchmarking supervised graph anomaly detection. *Advances in Neural Information Processing Systems*, 36:29628–29653, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

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, 2025a.

Yili Wang, Yixin Liu, Xu Shen, Chenyu Li, Rui Miao, Kaize Ding, Ying Wang, Shirui Pan, and Xin Wang. Unifying unsupervised graph-level anomaly detection and out-of-distribution detection: A benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=g90RNzs8wX.

Zhaohui Geoffrey Wang. Agenttrace: Causal graph tracing for root cause analysis in deployed multi-agent systems. *arXiv preprint arXiv:2603.14688*, 2026.

Ziyue Wang, Yurui Dong, Fuwen Luo, Minyuan Ruan, Zhili Cheng, Chi Chen, Peng Li, and Yang Liu. Escapecraft: A 3d room escape environment for benchmarking complex multimodal reasoning ability. *arXiv preprint arXiv:2503.10042*, 2025c.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, et al. Livebench: A challenging, contamination-limited llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology, 1971.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations. In *First Conference on Language Modeling*, August 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024a.

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.

Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. Benchmarking benchmark leakage in large language models. *arXiv preprint arXiv:2404.18824*, 2024b.

Wenkai Yang, Xiaohan Bi, Yankai Lin, Sishuo Chen, Jie Zhou, and Xu Sun. Watch out for your agents! investigating backdoor threats to llm-based agents. *arXiv preprint arXiv:2402.11208*, 2024.

Ziyi Yang, Iman Soltani, and Eric Darve. Anomaly detection with domain adaptation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2958–2967, 2023. doi: 10.1109/CVPRW59228.2023.00297.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng YAN. Agentracer: Who is inducing failure in the LLM agentic systems? In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=l05DseqvuD.

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 *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=GazlTYxZss.

Zhenliang Zhang, Xinyu Hu, Huixuan Zhang, Junzhe Zhang, and Xiaojun Wan. ICR probe: Tracking hidden state dynamics for reliable hallucination detection in LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17986–18002, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.880. URL https://aclanthology.org/2025.acl-long.880/.

Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019. URL http://jmlr.org/papers/v20/19-011.html.

Pengfei Zhou, Jiaxin Ai, Wangbo Zhao, Kai Wang, Xiaojiang Peng, Wenqi Shao, Hongxun Yao, Kaipeng Zhang, et al. Mpbench: A comprehensive multimodal reasoning benchmark for process errors identification. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21586–21606, 2025.

Shide Zhou, Kailong Wang, Ling Shi, and Haoyu Wang. Exposing the ghost in the transformer: Abnormal detection for large language models via hidden state forensics. *IEEE Transactions on Information Forensics and Security*, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*, 2023.

A Related Work

MAS Failure Analysis and Robustness. A growing body of work studies MAS reliability (Guo et al., 2024) through two complementary lenses. The first focuses on failure analysis: collecting execution traces from deployed systems and annotating them to understand why failures occur (Cemri et al., 2025; Deshpande et al., 2025; Zhang et al., 2025a; 2026). MAST (Cemri et al., 2025) establishes the most comprehensive failure taxonomy to date, covering 14 failure modes across 1,600+ traces from 7 frameworks. TRAIL (Deshpande et al., 2025) focuses on error categorization and span-level localization within agentic traces. Who&When (Zhang et al., 2025a) further attributes failures to specific agents and decisive error steps. AgenTracer (Zhang et al., 2026) automates trace annotation through counterfactual replay and programmatic fault injection. MP-Bench (Zhou et al., 2025) revisits failure attribution from a multi-perspective standpoint, showing that MAS failures often admit multiple plausible attributions due to complex inter-agent dependencies and ambiguous execution trajectories. Silent Failures (Pathak et al., 2025) is closest in spirit to MAS AD, curating two labeled datasets for anomaly detection, but the datasets have not been publicly released. The second lens focuses on robustness and security, including defenses against adversarial perturbations and agent hijacking (Wang et al., 2025a; Miao et al., 2025; Yang et al., 2024). While both lines of work provide valuable insights, they focus on post-hoc analysis or targeted defense rather than providing long-lived benchmark infrastructure for systematic MAS AD evaluation.

Anomaly Detection Benchmarks. In the broader AD literature, benchmark design has matured significantly across multiple data modalities. ADBench (Han et al., 2022) provides the most comprehensive tabular AD benchmark, examining 30 algorithms across 57 datasets along three angles: supervision level, anomaly type, and robustness to data corruption. TSB-AD (Liu & Paparrizos, 2024) addresses systematic flaws in time-series AD benchmarking, including dataset quality, evaluation metric bias, and hyperparameter tuning. GADBench (Tang et al., 2023) benchmarks supervised graph anomaly detection, comparing 29 models across 10 real-world datasets and finding that simple tree ensembles with neighborhood aggregation can outperform specialized GNNs. Complementing GADBench's supervised graph AD arena, UB-GOLD (Wang et al., 2025b) unifies unsupervised graph-level AD and OOD detection under one benchmark, exemplifying the shift toward unified evaluation across supervision regimes. Together, these works demonstrate the importance of structured benchmark angles, accurate labeling, and careful dataset curation for drawing meaningful conclusions from AD experiments. MAADBench is inspired by this design philosophy but addresses challenges unique to the MAS setting: rather than curating existing datasets, we generate traces from a live MAS framework, and rather than relying on human or algorithmic annotation, we obtain step-level labels through deterministic oracle verification.

LLM Agent Evaluation and Step-level Supervision. A large body of work constructs task environments and benchmarks for evaluating LLM agent capabilities across diverse domains. AgentBench (Liu et al., 2024) is the first systematic multi-environment benchmark for LLM-as-agent, covering 8 environments including OS, database, web browsing, and games. SWE-bench (Jimenez et al., 2023) evaluates agents on real GitHub issues; WebArena (Zhou et al., 2023) and GAIA (Mialon et al., 2023) test open-ended web navigation and general assistant capabilities; OSWorld (Xie et al., 2024) benchmarks multimodal agents on real computer environments; AppWorld (Trivedi et al., 2024) tests interactive coding agents in a controllable world of apps; and TheAgentCompany (Xu et al., 2024a) evaluates agents on consequential real-world professional tasks. Comprehensive surveys on LLM agent evaluation further systematize these efforts (Mohammadi et al., 2025). Despite their breadth, these benchmarks and surveys focus primarily on task completion rather than behavioral anomaly detection, and their verification mechanisms: unit tests, human judgment, or execution-based scripts, do not scale well to step-level labeling of agent traces. A complementary line of work has demonstrated the value of fine-grained process supervision: PRM800K (Lightman et al., 2023) and Math-Shepherd (Wang et al., 2024) show that step-level labels provide substantially better learning signal than the sparse outcome-level supervision alone, motivating the need for step-level ground truth in MAADBench. Concurrent work MM-Escape (Wang et al., 2025c) independently adopts escape rooms as an LLM evaluation environment, focusing on multimodal perception in a single-agent setting. In contrast, MAADBench instantiates escape rooms specifically for MAS anomaly detection, with multi-agent collaboration, generative task construction, and deterministic oracle-based step-level labeling as first-class design principles.

Benchmark Contamination and Long-lived Benchmarks. Benchmark contamination— test data leaks into LLM training corpora and inflates evaluation scores, has become a central concern in LLM evaluation (Magar & Schwartz, 2022; Golchin & Surdeanu, 2023; Deng et al., 2024a;b; Xu et al., 2024b). The field has responded by shifting from static to dynamic benchmarking paradigms (Chen et al., 2025). One line of work mitigates contamination by continuously sourcing questions from recent information: LiveBench (White et al., 2024) releases new questions monthly from recent math competitions, arXiv papers, and news articles, while LiveCodeBench (Jain et al., 2025) continuously collects new coding problems from online platforms. A complementary approach uses symbolic templates and parameter sampling to generate novel instances on demand: GSM-Symbolic (Mirzadeh et al., 2025) generates diverse math questions from symbolic templates, demonstrating that LLM performance on fixed benchmarks may reflect memorization rather than genuine reasoning. MAADBENCH adopts the template-based paradigm but extends it to the MAS AD setting: rather than sampling numerical parameters into fixed question templates, we compose entire task instances from parameterizable puzzle subtasks with hidden dependency structure, yielding a large combinatorial instance space that makes contamination practically infeasible.

B Discussion and Limitations

Benchmark Refresh and Anomaly Drift. A long-lived benchmark should not assume that anomaly distributions remain stationary as MAS components evolve. New LLM backbones, decoding configurations, or system components may induce different failure patterns, causing detectors trained on earlier traces to degrade. The value of MAADBENCH is therefore not to provide a permanently fixed dataset, but to support reproducible regeneration of labeled traces under new system configurations. This opens an important direction toward studying anomaly drift, detector transferability, and continual adaptation as MAS evolve.

Enabling MAS Anomaly Detection Benchmarking. The primary goal of MAADBENCH is to *enable* systematic MAS anomaly detection, rather than to exhaustively cover the MAS design space. Today, MAS architectures vary widely in their workflows, agent organizations, communication protocols, tools, and memory mechanisms, while standardized practices for constructing and evaluating MAS AD benchmarks remain largely absent. We therefore first establish a set of benchmark-design principles: reproducible trace generation, configurable task construction, and deterministic fine-grained labeling, that make repeated AD development and evaluation possible. In this sense, MAADBENCH serves as a foundation for defining how MAS AD benchmarks can be constructed and maintained as models and anomaly distributions evolve.

Expanding the MAS Design Space. Broader MAS configurations represent different instantiations of the same benchmark-design principles. Changes in harness, workflow, agent organization, communication, tools, or memory may expose new anomaly surfaces and propagation patterns, but the principles of configurable task construction, oracle-verifiable behavior, refreshable traces, and deterministic labeling remain unchanged. Extending MAADBENCH therefore amounts to defining a new MAS instantiation under the same paradigm, with its corresponding execution structure, anomaly granularity, and labeling oracles. An important future direction is to make MAADBENCH more modular and automated, enabling new MAS components and design choices to be incorporated with minimal effort while automatically generating the corresponding workflows, oracles, and labels.

C Labeling Cost Analysis

We compare the annotation burden of existing trace analysis benchmarks in Table 1 on a common per-trace basis, using a representative hourly rate of $H = \$20$/hr where applicable. Table 5 summarizes the estimated cost for each benchmark.

TRAIL (Deshpande et al., 2025). Annotation follows a five-person human pipeline: one primary annotator spends an average of 35 minutes per trace (30 min for GAIA, 40 min for SWE-Bench), followed by four independent verification rounds of 20 minutes each—totalling approximately 115 person-minutes per trace (Deshpande et al., 2025). At $H = \$20$/hr, this yields an estimated cost of **$38 per trace**.

Table 5: Estimated per-trace labeling cost across representative trace analysis benchmarks. $H =$ \$20/hr is used for human annotation estimates.

Benchmark	Annotation method	Est. cost/trace
TRAIL (Deshpande et al., 2025)	Human, 5-person pipeline	$\approx$ \$38
MAST (Cemri et al., 2025)	LLM-as-judge (`o1`) + human QA	$>$\$1
Who&When (Zhang et al., 2025a)	3 experts, 3-round consensus	$\approx$ \$9.2
Silent Failures* (Pathak et al., 2025)	Script + human GT per prompt†	small (unquantified)
AgenTracer (Zhang et al., 2026)	Counterfactual replay‡	unreported
MAADBENCH (ours)	Deterministic, generated ground-truth	$\approx$ **\$0**

† Partially automated: cycles/errors are detected by script, but drift still requires human inspection. ‡ Fully automated via counterfactual replay; only LLM API cost is incurred.

MAST (Cemri et al., 2025). Failure modes are labeled by an LLM-as-a-Judge pipeline using OpenAI `o1`, at a reported average API cost of \$1.80 per trace (ranging from \$0.37 to \$4.14 depending on the MAS framework) (Cemri et al., 2025). A subset of traces additionally receives human annotation (HE and HA) for quality assurance, though the corresponding human effort is not reported separately. The total per-trace cost therefore **exceeds \$1**, with the human overhead unquantified.

Who&When (Zhang et al., 2025a). Three expert annotators independently label every trace across three consensus rounds (independent labeling $\rightarrow$ uncertainty discussion $\rightarrow$ cross-validation). The total reported annotation effort is 84.3 person-hours for 184 traces (Zhang et al., 2025a), corresponding to approximately 27.5 minutes per trace per annotator. At $H =$ \$20/hr, this yields an estimated cost of **\$9.2 per trace**.

Silent Failures (Pathak et al., 2025). Cycle and error labels are assigned automatically by a rule-based script at negligible cost. Drift labels require a domain expert to define the ground-truth expected trajectory once per unique input prompt, after which labels are automatically propagated to all traces sharing that prompt (Pathak et al., 2025). Across 637 distinct prompts and 5,169 traces ($r \approx 8.1$ traces per prompt), this amortization substantially reduces the per-trace human burden, though the exact annotation time per prompt is not reported. The per-trace cost is **small but not precisely quantifiable**.

AgenTracer (Zhang et al., 2026). Labels are generated fully automatically via counterfactual replay and programmatic fault injection, requiring no human annotator (Zhang et al., 2026). An analyzer LLM (DeepSeek-R1) is invoked once per candidate step until the decisive error is identified, so the API cost scales with the depth of the error in the trace. No aggregate cost figure is reported by the authors; the per-trace cost depends on trace length and the LLM backbone used, but is **bounded by LLM API usage**.

MAADBENCH (ours). Ground-truth answers and labels are generated together with the puzzles and actions, requiring no human annotator or LLM judge. The marginal cost of labeling a new trace is therefore resulting in **effectively \$0** marginal cost per trace after environment construction.

D ESCAPE ROOM DESIGN DETAILS

D.1 AN EXAMPLE OF ESCAPE ROOMS.

We show an example of an easy escape room with one puzzle from the LIVECODEBENCH domain. The room contains three scenery items (Items 1–3: a concrete wall, an oak table, and a Persian rug) that serve as distractors with no task relevance. Item 4 is the *real clue*: it contains a chain of two syntactically valid Python programs and a hint identifying a thermometer measured on the absolute temperature scale, i.e., Kelvin. Item 7 is a *fake clue*: its first Python program is broken because the expression `offers[i][2)` has mismatched brackets and therefore would not run. Item 5 is the *real instrument*, a glass thermometer reading `88 celsius`. Item 6 is a *fake instrument*: although it is described as being in a comfortable room, it reads `-218 celsius`, which is inconsistent with the room state and should be discarded.

An escape room (easy) with one LiveCodeBench puzzle

```
You now see the following objects in the room:
  [Item 1] A plain concrete wall. There are a few hairline cracks
  ↪  running across it, but nothing you can pry open.
  [Item 2] A sturdy oak table in the center of the room. Its
  ↪  surface is scratched with faint marks that might once have
  ↪  meant something.
  [Item 3] A faded Persian rug covering most of the floor. Lifting
  ↪  the corner reveals nothing but dust underneath.
  [Item 4] A note on the floor which may contains a clue. It reads
  ↪  "This clue is a chain of two Python programs; run them in
  ↪  order.

FIRST program  run it on its call and let a = the value it returns:

def maxSubarrays(nums: List[int]) -> int:
    n = len(nums)
    mn = nums[0]
    for num in nums:
        mn &= num
    if mn:
        return 1
    res, cur = 0, nums[0]
    for i in range(1, n):
        if cur == mn:
            res += 1
            cur = nums[i]
        cur &= nums[i]
    if cur == mn:
        res += 1
    return res

Call: maxSubarrays(nums = [1, 0, 2, 0, 1, 2])

SECOND program  replace the argument `z` in its call with a (the
↪  value from the FIRST program), then run it. The value it
↪  returns is the clue's result:

def longestString(x: int, y: int, z: int) -> int:
    if x > y:
        return 2 * (y + min(y + 1, x) + z)
    else:
        return 2 * (x + min(x + 1, y) + z)

Call: longestString(x=3, y=2, z=a)"
Hint: Something with a thin glass tube and silver liquid awaits the
↪  answer, counted in the absolute scale that starts at nothing.
  [Item 5] A glass thermometer that measures temperature. It reads
  ↪  88 celsius.
  [Item 6] A glass thermometer mounted on the wall. The room feels
  ↪  perfectly comfortable. It reads -218 celsius.
  [Item 7] A note on the floor which may contains a clue. It reads
  ↪  "This clue is a chain of two Python programs; run them in
  ↪  order.

FIRST program  run it on its call and let a = the value it returns:

def maximizeTheProfit(n: int, offers: List[List[int]]) -> int:
    dp = [0] * (n + 1)
    offers.sort()
    i = 0
    for r in range(n + 1):
        dp[r] = max(dp[r], dp[r - 1])
```

```
        while i < len(offers) and offers[i][0] <= r:
            dp[offers[i][1] + 1] = max(dp[offers[i][1] + 1],
            ↪  dp[offers[i][0]] + offers[i][2)
            i += 1
    return dp[-1]

Call: maximizeTheProfit(n = 5, offers = [[0, 0, 1], [0, 2, 10], [1,
↪  3, 2]])

SECOND program  replace the argument `k` in its call with a (the
↪  value from the FIRST program), then run it. The value it
↪  returns is the clue's result:

def maximumBeauty(nums: List[int], k: int) -> int:
    nums.sort()
    j = 0
    ans = 0
    for i in range(len(nums)):
        while j < len(nums) and nums[j] - nums[i] <= 2 * k:
            j += 1
        ans = max(ans, j - i)
    return ans

Call: maximumBeauty(nums=[4, 6, 1, 2], k=a)
Hint: An instrument of orientation depends on what you find,
↪  counted in the unit where one means a full circle."
```

To solve this puzzle, one must first identify Item 4 as the real clue by checking code validity: both programs in Item 4 are runnable, whereas Item 7 contains a syntax error in its first program. The first program in Item 4 returns $a = 3$ on the given call. Substituting this into the second program gives `longestString(x=3, y=2, z=3)`, which returns 16. Following the benchmark instruction, the raw code result is transformed by adding $1{,}000{,}000$, yielding a final delta of $1{,}000{,}016$ Kelvin. Item 5 is then identified as the correct instrument. Since a temperature delta of $1{,}000{,}016$ Kelvin is equal to a delta of $1{,}000{,}016$ Celsius, applying it to the thermometer reading of `88 celsius` yields $88 + 1{,}000{,}016 = 1{,}000{,}104$ Celsius. The puzzle is solved with the correct adjusted reading, and the room state reflects this by removing the clue and the instrument. A more detailed action and labeling description is provided in Appendix D.5.

D.2 Puzzle Generation

Instruments. Each escape room instance is procedurally generated as a set of interconnected puzzles, where each puzzle is anchored by one of four physical instrument types: *thermometer* ($[-20, 120]°$, convertible to units of either Celsius, Fahrenheit, or Kelvin; *compass* ($[0, 359]°$; convertible to units including degree, radians or turns), *clock* ($[0, 86399]$ s; convertible to seconds, minutes or hours, with 24-hour wrapping), and *scale* ($[1, 100]$ kg; convertible to grams, pounds, or ounces).

Hint. For each puzzle, we sample a math/coding word problem from the GSM-Hard dataset (Gao et al., 2023) or LiveCodeBench-execution (Jain et al., 2025) as the *clue*, whose numerical answer serves as a delta to be applied to the instrument's current reading after appropriate unit conversion. The ground-truth answer is thus $s' = \text{wrap}(s + \Delta_{\text{converted}})$, where s is the instrument's current state and $\Delta_{\text{converted}}$ is the clue answer converted into the instrument's native unit. A poetic hint (e.g., "*the scale where water freezes at zero*") links each clue to its target instrument without revealing the answer directly.

Traps and Distractions. To increase difficulty, we introduce three layers of *traps*: (1) **fake clues** ($n_{\text{fake}} \in [1, 5]$), which contain misleading hints referencing non-existent instruments; (2) **fake items** ($n_{\text{fake}} \in [1, 5]$), which share the same instrument type as the real item but exhibit telltale signs such as out-of-range readings or descriptions of broken/non-functional states (e.g., "*the mercury level hasn't shifted in hours*"); and (3) **item-value description traps**, which embed adversarial text in the real item's description, suggesting that the current reading is already the correct answer. Additionally, n_{scenery} irrelevant decorative objects are placed in the room as distractors.

D.3 EXTREME ROOM DESIGN

Overall. Extreme rooms extend the standard EscapeRoom design with explicit inter-puzzle dependencies, requiring agents to determine a valid solving order. Each room contains four puzzles, one per instrument; solving a puzzle unlocks some instruments and locks others, so the agent must plan the order such that all four puzzles eventually become solvable. We formalise this as a STRIPS planning instance (Fikes & Nilsson, 1971), a classical framework also underlying the Blocks World domain (Winograd, 1971).

A Concrete Example. A Extreme room with four puzzles (P_{CLOCK}, $P_{\text{THERMOMETER}}$, P_{COMPASS}, P_{SCALE}) starts with three instruments unlocked (Thermometer, Compass, Scale) and Clock locked. Solving each puzzle modifies the lock states of others; only *two* valid orderings exist:

- Path A: $P_{\text{COMPASS}} \to P_{\text{SCALE}} \to P_{\text{THERMOMETER}} \to P_{\text{CLOCK}}$
- Path B: $P_{\text{COMPASS}} \to P_{\text{CLOCK}} \to P_{\text{SCALE}} \to P_{\text{THERMOMETER}}$

Both require starting with P_{COMPASS}. Four other orderings lead to *deadends*, with the deepest making three seemingly valid moves before the room becomes unsolvable. The planner is given full state information at every step, with nothing hidden. A globally reasoning planner can trace the consequences of each choice and identify P_{COMPASS} as the only safe first move; a greedy, one-step-lookahead planner naturally falls into deadends—for example, choosing P_{SCALE} at step 0 looks harmless (unlocks Thermometer, locks nothing) but creates a downstream constraint that eventually makes the room unsolvable. The deadend emerges *organically* from short-horizon planning against a DAG requiring global foresight.

DAG Sampling. The lock/unlock effects induce a directed dependency graph over puzzles. We enumerate all $4! = 24$ possible orderings via depth-first search over the state space. A *deadend* occurs when all remaining instruments are locked before the goal is reached. A *trap first move* is a first-step choice after which no solution path exists regardless of subsequent decisions.

Extreme DAGs are generated by rejection sampling and can be configured with different difficulty levels. The current configuration is: (1) at least one complete solution path exists; (2) at least two deadend paths exist; (3) at least one deadend path has depth ≥ 2, ensuring traps are not trivially detectable at the first step; and (4) at least two first-step choices are guaranteed traps, with at least one safe choice remaining — bounding the random success rate at the first step to at most $1/3$. These constraints are verified exactly by exhaustive graph enumeration, which is tractable given the fixed state space of $|\mathcal{L}| \leq 4$ instruments and four puzzles. The full configuration space of $4 \times 15^4 = 202{,}500$ possible DAGs yields **2,376** structurally valid instances satisfying all constraints (Appendix D.4).

D.4 TASK INSTANCE SPACE

We calculate the task instance space with GSM-hard math questions as an example. Each puzzle instance is determined by the independent combination of the following factors: the math questions (1,319 choices); the instrument type (one of four: clock, compass, thermometer, or scale); the initial instrument reading ($|\mathcal{S}_i|$, ranging from 100 to 86,400 states depending on the instrument); and the delta unit ($|\mathcal{U}_i|$, 3–4 choices per instrument type). For Extreme rooms, an additional dependency layer contributes 2,376 structurally distinct DAG configurations (Appendix D.3). These factors only cover the components required to unlock each puzzle/room; we conservatively exclude additional variation from distractors and traps.

Since each puzzle in a room is configured independently, the total instance space is the product of per-puzzle factors across all n_d puzzles: $\mathcal{I}(d) = (|\mathcal{P}| \cdot |\mathcal{S}_i| \cdot |\mathcal{U}_i|)^{n_d}$, where $|\mathcal{P}| = 1{,}319$. Because the instrument type varies per puzzle, the instance space is bounded by the least flexible instrument (Scale: $100 \times 4 = 400$ configurations) and the most flexible (Clock: $86{,}400 \times 3 = 259{,}200$ configurations), giving a per-puzzle base factor between $1{,}319 \times 400 \approx 5 \times 10^5$ and $1{,}319 \times 259{,}200 \approx 3.4 \times 10^8$. The resulting instance space ranges are:

Easy ($n = 1$): $[5 \times 10^5,\ 3.4 \times 10^8] \approx [10^6,\ 10^9]$

Medium ($n = 2$): $[(5 \times 10^5)^2,\ (3.4 \times 10^8)^2] \approx [10^{11},\ 10^{17}]$

Table 6: Task instance space per difficulty level. Each range reflects the least flexible instrument (Scale, 100×4 configurations) as the lower bound and the most flexible (Clock, $86{,}400 \times 3$) as the upper bound. Extreme additionally multiplies by 2,376 valid DAG configurations.

Difficulty	Puzzles	Fake Items/Clues	DAG Variants	Instance Space
Easy	1	1	—	$[10^6,\ 10^9]$
Medium	2	3	—	$[10^{11},\ 10^{17}]$
Hard	3	5	—	$[10^{17},\ 10^{26}]$
Extreme	4	3	2,376	$[10^{26},\ 10^{37}]$

Hard ($n = 3$): $[(5 \times 10^5)^3,\ (3.4 \times 10^8)^3] \approx [10^{17},\ 10^{26}]$

Extreme ($n = 4$, $\times$ 2,376 DAG variants): $[(5 \times 10^5)^4 \times 2376,\ (3.4 \times 10^8)^4 \times 2376] \approx [10^{26},\ 10^{37}]$

Table 6 summarises these ranges across all difficulty levels.

The combinatorial scale across all difficulty levels renders memorisation-based solutions intractable and ensures that each of the evaluation traces constitutes a statistically independent problem instance.

D.5 Action Labeling

Each action is labeled as *success* or *failure* by comparing the agent's structured output with the pre-generated ground truth. For failed actions, an error-code is further assigned during labeling. The action-specific labeling criteria are as follows.

SELECT_PUZZLE (Extreme only). The agent must select a puzzle that is currently available (i.e., its instrument is unlocked and the puzzle has not yet been solved). The action is labeled a failure if the selected puzzle has already been solved (error-code: `already_solved`), if its instrument is currently locked (`puzzle_locked`), if the puzzle ID does not exist in the room (`puzzle_not_found`), or if the selection leads to an irrecoverable deadend state (`leads_to_deadend`).

OBSERVE_CLUE. The agent must select the item ID corresponding to the real clue. The action is labeled a failure if the selected item ID does not match the ground-truth clue ID (`clue_select_wrong`).

SOLVE_CLUE. The agent must produce both the correct numeric answer and the correct delta unit. The action is labeled a failure if the numeric value deviates from the ground-truth answer beyond a relative tolerance of 10^{-4} (`math_value_wrong`), or if the reported unit does not match the ground-truth unit (`math_unit_wrong`). Both checks are applied independently.

OBSERVE_INSTRUMENT. Given an instrument type from SOLVE_CLUE, the agent must identify the correct instrument from the distractors. It also needs to report its current reading. The action is labeled a failure if the reported instrument ID does not match the ground-truth ID (`instrument_wrong`), or if the reported state deviates from the ground-truth reading beyond a relative tolerance of 10^{-4} (`instrument_state_wrong`).

APPLY_DELTA. The agent must compute the correct final reading after applying the delta under the correct unit conversion. The action is labeled a failure if the final answer does not match the ground-truth result (`apply_delta_wrong`).

TOOL_CALL (Tool-enabled setting only). When unit-conversion tools are enabled and a conversion is required, the agent must issue a valid structured tool call before producing the final APPLY_DELTA answer. The action is labeled a failure if the required conversion is not called (`TC_REQUIRED_NOT_CALLED`), if the selected tool is incompatible with the required conversion (`TC_WRONG_TOOL`), if the tool arguments use an incorrect value or source/target unit (`TC_WRONG_ARGUMENTS`), if the requested conversion crosses incompatible unit dimensions (`TC_DIMENSION_MISMATCH`), if the tool call is malformed or otherwise invalid (`TC_INVALID_REQUEST`), or if the agent omits the required auditable tool-call message (`TC_MISSING_MESSAGE`). These tool-call failures are counted as action-level anomalies; in the failure-mode analysis, their downstream effect is captured through the parent APPLY_DELTA action.

OBSERVE_PUZZLE. The agent must correctly assess whether the puzzle has been solved based on the current room state. The action is labeled a failure if the agent's judgment disagrees with the actual puzzle state (`observer_disagrees`).

E MAS DESIGN DETAILS

E.1 MAS HARNESS

Our harness implements a sequential, role-based multi-agent workflow comprising an observer, a clue solver, an item manager, and a planner for rooms with puzzle dependencies. Each role can perform multiple related actions. The orchestrator constructs action-specific prompts from the current environment state and upstream messages. Agents return JSON objects containing a natural-language `message` and `structured` outputs; these outputs are passed to downstream agents together with the message as a `reasoning` field. Submitted instrument readings trigger environment updates, which inform subsequent observations and planning decisions. The harness records action inputs and outputs, agent identities, LLM-call token usage and latency, and tool-call metadata, and annotates actions against the room's oracle information. This scaffold provides a controlled setting for evaluating action-level reasoning and error propagation.

Tool calls. The harness supports both Tool and No-Tool settings under the same role structure and action pipeline. In the No-Tool setting, the agent performs unit conversion itself within `APPLY_DELTA`. In the Tool setting, it can request unit conversion tools through `convert_time`, `convert_temperature_delta`, `convert_angle`, and `convert_mass`. A request specifies the tool name and the arguments `value`, `from_unit`, and `to_unit`. The harness returns the result or error to the agent, which can issue another request, up to three calls, or submit its final answer. Oracle-based checks identify missing required calls, incorrect tool choices or arguments, and incorrect final readings.

E.2 ACTION AND AGENT PROMPTS

We provide representative prompts here for reference; the complete prompt templates are available in our code repository.

Agent Prompts

```
Observer:
  You are the Observer in an escape room. You examine objects in the
  room and distinguish real instruments and clues from fakes.
  Fake items have tells: broken/damaged descriptions, or readings that
  are physically impossible for what they claim to be.
  Fake clues reference nonsensical instruments or units that don't
  exist. Real clues contain a solvable math problem and a hint
  pointing to a specific type of instrument and unit.
  Always respond with a single valid JSON object -- no markdown,
  no extra text.
Clue Solver:
  You are the Clue Solver. You receive a math word problem extracted
  from a clue found in the room. Solve it step by step and return
  the numerical answer.
  Always respond with a single valid JSON object -- no markdown,
  no extra text.
Item Manager:
  You are the Item Manager. You receive an instrument's current
  reading, a delta value, and a unit. Compute the new reading after
  applying the delta. Pay attention to unit conversions.
  Always respond with a single valid JSON object -- no markdown,
  no extra text.
Planner:
  You are the Planner in an escape room with multiple puzzles.
  You must choose which puzzle to solve next based on item lock
```

```
  states and DAG effects. A wrong ordering may permanently lock
  required items.
  Always respond with a single valid JSON object -- no markdown,
  no extra text.
```

Action: APPLY_DELTA (agent: item_manager)

```
You are operating an instrument in an escape room.

[Previous input -- OBSERVE_INSTRU structured JSON]:
  "item_type" -- instrument type
  "item_state" -- current reading of the instrument (number)
  "item_unit" -- unit of the current reading
  "delta" -- the value to add to the reading (from the clue solver)
  "delta_unit" -- the unit of the delta
IMPORTANT: use ONLY the values given in the JSON. Do not substitute
or guess.

Your task: compute the new instrument reading after applying the delta.

Step-by-step instructions:
  1. Extract the instrument's current reading, unit, delta value,
     and delta unit from the input above
  2. If the delta unit differs from the instrument's unit, convert
     the delta first
  3. Add the converted delta to the current reading
  4. Apply wrapping if needed (clocks wrap at 24h, compasses
     at 360 degrees)

Return a JSON object with exactly these two keys:

"message": "<step-by-step calculation showing unit conversion and
            final result>",
"structured": {
  "answer": <final reading. Clocks: "HH:MM:SS"; others: integer>,
  "converted_delta": <delta after unit conversion, in instrument's
                      unit>,
  "converted_unit": "<unit after conversion -- same as instrument
                      unit>"
}
```

Action: OBSERVE_CLUE (agent: observer)

```
You are observing an escape room. Here is everything you can see:

{room_desc}

Your task: identify the ONE real clue among the notes on the floor.

A REAL clue has both:
  1. A math problem (a question involving numbers and calculation)
  2. A hint that describes a real instrument and a unit of measurement

Select the single real clue.

Return a JSON object with exactly these two keys:

"message": "<Include the FULL TEXT of the math problem and the FULL
            TEXT of the hint. The next agent needs the exact problem
            to solve it.>",
"structured": {
```

```
  "item_id": <integer, the item id of the clue>,
  "clue":    "<exact full text of the clue>"
}
```

Action: OBSERVE_INSTRU (agent: observer)

```
You are observing an escape room. Here is everything you can see:

{room_desc}

[Previous input -- SOLVE_CLUE structured JSON]:
  "answer" -- the numerical delta value computed from the clue
  "unit"   -- the unit of that delta

Your task: identify the ONE real instrument that the clue's hint
refers to.
A REAL instrument:
  - Has a physically valid reading and a working description
  - Matches the instrument type described in the clue
Select the single real instrument.

Return a JSON object with exactly these two keys:
"message": "<Include: (1) the instrument you found and its current
            reading with unit, (2) the delta value and delta unit
            from the solver's input above.>",
"structured": {
  "item_index": <integer, 1-based index of the instrument>,
  "item_type":  "<one of: {valid_types}>",
  "item_state": <current reading as a number>,
  "item_unit":  "<unit of the current reading>",
  "delta":      <number -- the delta value from the solver>,
  "delta_unit": "<unit of the delta from the solver>"
}
```

Action: OBSERVE_PUZZLE (agent: observer)

```
You are observing an escape room. Here is everything you can see:

{room_desc}

[Previous input -- OBSERVE_CLUE and OBSERVE_INSTUR structured JSONs]

Your task: determine whether the current puzzle has been solved by
comparing the current room contents against your previous observation.

Step 1 -- Inventory changes:
  List every item that DISAPPEARED and every item that is NEW.
Step 2 -- Check the clue and instrument you previously identified:
  - Is the clue you identified still present in the room?
  - Is the instrument you identified still present in the room?
Step 3 -- Determine solved:
  The puzzle is solved ONLY if BOTH the previously identified clue
  AND instrument have DISAPPEARED. If either is still present,
  the puzzle is NOT solved.

Return a JSON object with exactly these two keys:

"message": "<Step 1: list disappeared/new items. Step 2: check
            presence. Step 3: conclusion.>",
"structured": {
  "solved":       <true ONLY if both clue and item are gone>,
```

```
  "clue_present": <true if the previously identified clue is visible>,
  "item_present": <true if the previously identified instrument is
                   visible>
}
```

Action: SELECT_PUZZLE (agent: planner)

```
You are a planner in an escape room with multiple puzzles.
Each puzzle is tied to an item. Items can be UNLOCKED or LOCKED.
You can only solve puzzles whose item is UNLOCKED.
Solving a puzzle may unlock or lock other items -- choose wisely.

{planning_desc}

Your task: choose ONE puzzle to solve next.
Consider the unlock/lock effects carefully -- a wrong ordering
may permanently lock required items (deadend).

Return a JSON object with exactly these two keys:

"message": "<your reasoning about which puzzle to solve and why>",
"structured": {
  "puzzle_id": "<the puzzle_id you choose to solve next>",
  "item_type": "<item type of the chosen puzzle --
                one of: {valid_types}>"
}
```

Action: SOLVE_CLUE (agent: clue_solver)

```
You are a clue solver in an escape room.

[Previous input -- OBSERVE_CLUE structured JSON]:
  "item_id" -- the clue's item id
  "clue" -- the full text of the clue (math problem + instrument hint)

Your task: extract the math problem from the input above and solve it.
Your numerical answer will later be used as a delta with a specified
unit to adjust an instrument in the room.

Instructions:
  - Solve step by step
  - Your final answer must be a single number
  - The input above tells you which unit your answer should be
    expressed in -- include that unit in your structured output
  - Do NOT apply the delta to any instrument yet

Return a JSON object with exactly these two keys:

"message": "<step-by-step solution of the math problem,
             show all work>",
"structured": {
  "answer": <number -- the final solution>,
  "unit":   "<unit string -- must be one of: {valid_units}>"
}
```

E.3 Computational Cost of MAADBench-Full

Cost of Generating MAADBench-Full. Table 7 reports the API cost of generating MAADBench-Full across five SOTA LLM backbones, aggregated over our full sampling sweep (two clue domains – GSM-Hard and LiveCodeBench – three sampling temperatures $\{0.0, 0.3, 0.6\}$, and with/without the unit-conversion tool). The total cost is $394.32 for 5,200 traces (averaging $0.076 per trace). Cost

varies primarily with output token volume and price per million tokens: Claude Sonnet 4.6 ($146.63) is the most expensive backbone due to its $15/1M output rate combined with the highest aggregate token volume, while DeepSeek-R1 ($39.57) is the cheapest despite producing the most output tokens (13.05M), owing to its $0.55/$2.19 per-1M rate. Claude Opus 4.8 ($93.44) currently covers only the temperature-0.0 configurations (400 traces) and is thus not directly comparable on a per-backbone total, though at $0.234/trace it is the most expensive backbone per trace.

Table 7: API cost for generating MAADBENCH-FULL across five SOTA LLM backbones, aggregated over all sampled configurations (domains, temperatures, and tool/no-tool variants). Prices are based on official API rates at experiment time. Claude Opus 4.8 currently covers only the default temperature configuration (400 traces); all other backbones cover 1,200 traces each.

Model	Traces	API Calls	Input Tokens	Output Tokens	Price (in/out per 1M)	Total Cost
DeepSeek-R1	1,200	10,267	19.96M	13.05M	$0.55 / $2.19	$39.57
Claude Sonnet 4.6	1,200	10,329	20.26M	5.72M	$3.00 / $15.00	$146.63
Claude Opus 4.8	400	4,025	10.27M	1.68M	$5.00 / $25.00	$93.44
GPT-4.1	1,200	7,605	13.27M	2.81M	$2.00 / $8.00	$48.98
GPT-5.4	1,200	8,445	14.01M	2.04M	$2.50 / $15.00	$65.71
Total	**5,200**	**40,671**	**77.77M**	**25.31M**	–	**$394.32**

F ADDITIONAL ANOMALY PROFILING

Mapping MAADBENCH Anomalies to Failure Modes. To support interpretable anomaly analysis, MAADBENCH automatically maps each failed action (by their error code, Appendix D.5) to a MAST (Cemri et al., 2025) failure mode, through rule-based classification. Table 8 summarizes the rules.

Extending the MAST Taxonomy. MAST's original taxonomy (FC1–FC3) mainly covers system-level failures, including prompt design, agent coordination, and output verification. However, MAADBENCH reveals two additional failure families that are central to MAS AD. We therefore extend MAST with **FC4 (Agent Capability)** and **FC5 (Error Propagation)**. FC4 captures single-agent capability failures independent of orchestration, including **FM-4.1 Deception Susceptibility**, where an agent follows a distractor, and **FM-4.2 Reasoning Error**, where arithmetic or logical reasoning fails. **FM-5** captures downstream action failures due to upstream failures. These categories are orthogonal to the coordination and prompt-design failures targeted by MAST's FC1–FC3.

Figure 9 shows the resulting failure-mode distribution across LLM backbones. FM-4.2 and FM-5 dominate failures across all backbones, indicating that reasoning failures and error propagation are core failure patterns in MAADBENCH.

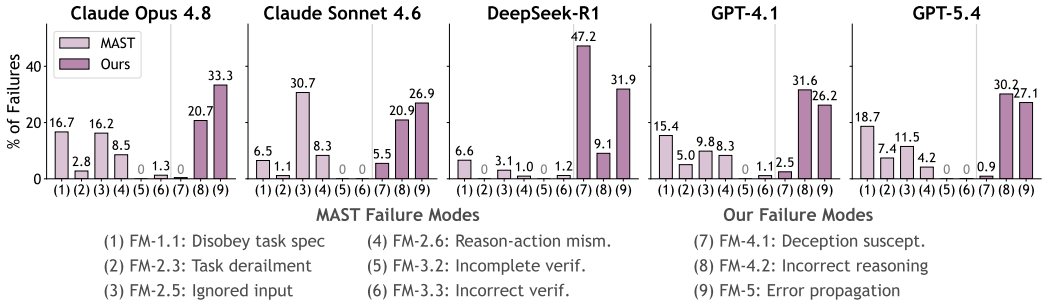


Figure 9: Distribution of failures across five LLM backbones, broken down by MAST failure modes (FM-1.1–FM-3.3, light bars) and MAADBENCH's extended failure modes (FM-4.1, FM-4.2 and FM-5, dark bars). FM-4.2 (Reasoning Error) and FM-5 (Error Propagation) dominate failures across all backbones.

Table 8: Deterministic mapping from MAADBENCH action-level error codes to MAST failure modes. Cascade rule: if any upstream action on the same puzzle already failed, the downstream failure is labeled as **FM-5 Error Propagation**.

Action	Error code	Failure mode
SELECT_PUZZLE	`puzzle_locked`	FM-1.1 Disobey task spec.
	`already_solved`	FM-1.3 Step repetition
	`leads_to_deadend`	FM-2.3 Task derailment
	`puzzle_not_found`	FM-2.6 Reason–action mismatch
OBSERVE_CLUE	`clue_select_wrong` (GT id $\in$ output reasoning content)	FM-2.6 Reason–action mismatch
	`clue_select_wrong` (GT id $\notin$ output reasoning content)	FM-4.1 Deception suscept.
SOLVE_CLUE	`math_unit_wrong` (value ok)	FM-1.1 Disobey task spec.
	`math_value_wrong`, GT $\in$ output reasoning content	FM-2.6 Reason–action mismatch
	`math_value_wrong`, GT $\notin$ output reasoning content	FM-4.2 Reasoning error
OBSERVE_INSTRUMENT	`instrument_state_wrong` (id ok)	FM-2.5 Ignored input
	`instrument_wrong`, GT $\in$ output reasoning content	FM-2.6 Reason–action mismatch
	`instrument_wrong`, GT $\notin$ output reasoning content	FM-4.1 Deception suscept.
TOOL_CALL	`TC_REQUIRED_NOT_CALLED`, `TC_DIMENSION_MISMATCH`, `TC_INVALID_REQUEST`	FM-1.1 Disobey task spec.
	`TC_WRONG_ARGUMENTS`, `TC_WRONG_TOOL`	FM-2.5 Ignored input
	`TC_MISSING_MESSAGE` only	FM-3.2 Incomplete verification
APPLY_DELTA	`apply_delta_wrong`, upstream values used	FM-4.2 Reasoning error
	`apply_delta_wrong`, upstream values absent	FM-2.5 Ignored input
OBSERVE_PUZZLE	`observer_disagrees`	FM-3.3 Incorrect verif.
any downstream	*cascade from upstream failure*	**FM-5 Error propagation**

G EXPERIMENTAL SETUP DETAILS

We organize the anomaly detection algorithms in MAADBENCH in the following four supervision levels.

G.1 DATA PROCESSING

For all methods, we randomly selected 80% of the actions for training using stratified sampling, with the remaining 20% reserved for testing. Each random seed produces a different combination of samples. The test set is consistent across all methods, while the training set differs depending on the level of supervision.

Tabular Methods. We convert each action trace into a tabular feature vector through the following steps. First, we clean raw LLM reasoning outputs by removing markdown artifacts, special characters, and normalizing whitespace. We then extract statistical features for each trace, including: (i) LLM call statistics (duration mean/min/max/std), and (ii) token consumption (input/output tokens mean/min/max/std). For semantic information, to ensure the previous context of an action, we concatenate all action outputs up to the current action and embed them using a pre-trained sentence transformer (all-MiniLM-L6-v2, 384-dim), as its contextual representation.

Graph-based Methods. For graph-based anomaly detection, we construct heterogeneous graphs from execution traces. We extract directed edges from the execution sequence: if Agent A is followed by Agent B, we add edge $A \rightarrow B$. Each agent node stores its embedded system prompt as a node attribute, while edges store embedded LLM reasoning outputs passed to the next agent, along with call duration metadata.

MAS-specific Methods. G-Safeguard and BlindGuard build directly on the graph representation described above.

LLM-based Detectors. We leverage a open-source LLM (Gemma-4) as an anomaly detector. The model receives complete execution traces in JSON format, with a detailed system prompt, motivated by MAST (Cemri et al., 2025), that provides task context, describes the multi-agent system components. Anomaly definition prompts are revised based on MAST (Cemri et al., 2025) LLM-as-a-judge prompts.

G.2 METHOD DETAILS

Unsupervised Methods. Require no labels for both normal and abnormal data.

1. **Isolation Forest** (Liu et al., 2008) is an ensemble-based anomaly detection method that isolates observations by randomly selecting a feature and splitting the data. Anomalies are identified as instances that require fewer splits to be isolated, resulting in shorter average path lengths in the ensemble of isolation trees.
2. **KNN** (Ramaswamy et al., 2000) detects anomalies by computing the distance of each data point to its k-nearest neighbors. Points with large distances to their neighbors are considered anomalous, as they lie in sparse regions of the feature space.
3. **LOF** (Breunig et al., 2000) (Local Outlier Factor) measures the local density deviation of a data point with respect to its neighbors. Anomalies are identified as points whose local density is significantly lower than that of their neighbors, indicating they reside in sparser regions.
4. **Gemma4-E2B (ZS)**, **Gemma4-E4B (ZS)**, **Gemma4-31B (ZS)**, and **Gemma4-26B-A4B (ZS)** (Google DeepMind, 2026) are open-source LLMs from the Gemma4 family, applied in a zero-shot setting. We choose Gemma for two reasons: (i) its open-source release supports reproducibility, and (ii) its multiple size variants (2B–31B parameters) enable us to systematically study how LLM-based detection scales with model capacity. They are prompted to assess whether a given instance deviates from normal behavior based purely on the model's pretrained knowledge. The prompts are provided here `https://drive.google.com/file/d/1MJyKp7X2xA_8PwSQOcKqc5MPCXZAyIkW/view?usp=sharing`.

One-Class Classification (OCC) Methods. These methods are trained exclusively on normal samples and detect anomalies as deviations from the learned normal distribution.

1. **OC-SVM** (Schölkopf et al., 2001) (One-Class Support Vector Machine) learns a decision boundary in a high-dimensional feature space that encloses the majority of normal training data. Test points falling outside this boundary are flagged as anomalies.
2. **Autoencoder** (Sakurada & Yairi, 2014) projects data to a lower-dimensional latent space and reconstructs it through an encoding-decoding architecture. Anomalies are characterized by high reconstruction errors, as the model fails to accurately reconstruct patterns unseen during training.
3. **DeepSVDD** (Ruff et al., 2018) trains a neural network to map normal data into a compact hypersphere in a latent space. Anomaly scores are computed as the distance of a test point's representation from the center of the hypersphere.
4. **TAM** (Qiao & Pang, 2023) learns node representations by maximizing local node affinity based on the one-class homophily property, where normal nodes exhibit stronger similarity to their neighbors. Anomalies are identified as nodes with low local affinity.
5. **DOMINANT** (Ding et al., 2019) leverages deep graph autoencoders to reconstruct both the graph structure and node attributes. Anomaly scores are derived from the joint reconstruction error of these two components.
6. **BlindGuard** (Miao et al., 2025) learns to detect anomalies via corruption-guided training, where pseudo anomalies are generated through feature perturbations and optimized using contrastive learning.

Semi-Supervised Methods. Require a small set of labeled abnormal data as the training set.

1. **DeepSAD** (Ruff et al., 2019) extends the Deep SVDD framework to the semi-supervised setting by incorporating a small number of labeled anomalies during training. The objective encourages normal samples to map close to the hypersphere center while pushing labeled anomalies away from it.
2. **DevNet** (Pang et al., 2019) is a semi-supervised anomaly detection method that leverages a small set of labeled anomalies to learn a deviation network. It optimizes an anomaly score function such that labeled anomalies receive significantly higher scores than normal samples.

3. **GGAD** (Qiao et al., 2024) exploits a limited number of labeled anomalies within a graph-based framework to guide the anomaly detection process. It propagates anomaly information through the graph structure to improve detection coverage.
4. **G-Safeguard (Semi-sup)** (Wang et al., 2025a) is the semi-supervised variant of G-Safeguard, where only a small subset of labeled anomalies is provided instead of the full set.
5. **Gemma4-E2B (FS)**, **Gemma4-E4B (FS)**, **Gemma4-31B (FS)**, and **Gemma4-26B-A4B (FS)** (Google DeepMind, 2026) are applied in a few-shot setting where 3 normal and 3 abnormal examples are provided in the prompt as context. This allows the model to better calibrate its notion of normality without any gradient-based training.

Supervised Methods. Require labeled normal and abnormal data as the training set.

1. **SVM** (Hearst et al., 1998) (Support Vector Machine) is a classical supervised classification method that finds a maximum-margin hyperplane to separate normal and anomalous samples in the feature space.
2. **Random Forest** (Breiman, 2001) is an ensemble learning method that constructs multiple decision trees during training and outputs the majority class prediction. It is robust to overfitting and performs well on high-dimensional tabular data.
3. **XGBoost** (Chen & Guestrin, 2016) is a gradient boosting framework that builds an ensemble of decision trees in a sequential manner, where each tree corrects the errors of the previous one. It is widely adopted for its strong performance on structured data.
4. **G-Safeguard** (Wang et al., 2025a) is a supervised graph-based anomaly detection method that models MAS as a graph and uses GNN-based anomaly detection on the multi-agent utterance topology to identify compromised agents.

Implementation Details. Tabular methods are implemented via Scikit-learn (Pedregosa et al., 2011) and PyOD (Zhao et al., 2019) with default hyperparameters; graph-based methods adapt author-provided codebases; and LLM-as-detector methods are motivated by MAST prompting (Cemri et al., 2025). All methods generate an anomaly score for each test sample. These scores are then converted into binary labels by thresholding based on the known number of anomalies in the test set, following prior work (Bergman & Hoshen, 2020; Shenkar & Wolf, 2022). LLM's use logits of a constrained single-token ("yes"/"no") output as anomaly scores.

AD Evaluation Resource Cost. The non-LLM detectors run on a single NVIDIA L40s GPU and complete within minutes per method (3–26 minute range), with tabular methods averaging 4.8 minutes, graph-based methods 9.0 minutes, and G-Safeguard 4.6 minutes, totaling under 2.4 hours. The LLM-based detectors run on a single NVIDIA A100 GPU. We benchmark four Gemma variants from 2B to 31B parameters; the largest (Gemma-4-31B) processes 69.94M tokens across 1,345 traces (33,000 tokens/trace under semi-supervised; 19,000 under unsupervised), taking 6.2 hours, while smaller variants (Gemma-4-E2B/E4B) take 2.5–3.0 hours each. Across all Gemma variants, total LLM-based benchmarking takes 14.0 hours.

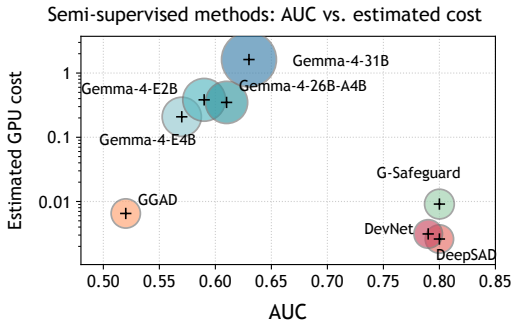


Figure 10: AUCROC vs. Normalized runtime GPU cost of semi-supervised methods.

Detection Performance vs. Runtime Cost. Figure 10 compares the detection performance and normalized runtime GPU cost of semi-supervised methods. LLM-based detectors incur orders-of-magnitude higher runtime cost while achieving only moderate AUC ($\sim$ 0.57–0.63). In contrast, several non-LLM methods, particularly DeepSAD, DevNet, and G-Safeguard, achieve substantially

higher detection performance ($\sim$ 0.79–0.80) at much lower cost. This large efficiency gap suggests that scaling LLM-based detectors alone is not a cost-effective path for MAS AD, motivating lightweight, hybrid, or selectively invoked LLM-based designs.

H Additional Evaluation Results

Table 9: Overall anomaly detection performance across four supervision settings. Dark blue cells indicate $p < 0.01$ and light blue cells indicate $p < 0.05$ relative to *Random* performance.

Supervision	Method	Method Family	F1	Acc	AUC	Bal. Acc
-	Random	Random	$.176_{\pm.021}$	$.705_{\pm.021}$	$.494_{\pm.013}$	$.498_{\pm.013}$
Unsupervised	Isolation Forest	Tabular	$.281_{\pm.041}$	$.743_{\pm.012}$	$.543_{\pm.039}$	$.562_{\pm.020}$
	KNN	Tabular	$.239_{\pm.032}$	$.728_{\pm.024}$	$.488_{\pm.038}$	$.537_{\pm.021}$
	LOF	Tabular	$.056_{\pm.009}$	$.662_{\pm.027}$	$.378_{\pm.026}$	$.425_{\pm.011}$
	Gemma4-E2B (zero-shot)	LLM	$.296_{\pm.013}$	$.746_{\pm.014}$	$.560_{\pm.014}$	$.571_{\pm.005}$
	Gemma4-E4B (zero-shot)	LLM	$.332_{\pm.027}$	$.758_{\pm.018}$	$.596_{\pm.020}$	$.593_{\pm.017}$
	Gemma4-26B-A4B (zero-shot)	LLM	$.244_{\pm.035}$	$.729_{\pm.013}$	$.542_{\pm.021}$	$.540_{\pm.017}$
	Gemma4-31B (zero-shot)	LLM	$.288_{\pm.033}$	$.741_{\pm.010}$	$.594_{\pm.014}$	$.564_{\pm.014}$
One-Class	Autoencoder	Tabular	$.296_{\pm.036}$	$.748_{\pm.020}$	$.511_{\pm.029}$	$.571_{\pm.021}$
	DeepSVDD	Tabular	$.245_{\pm.056}$	$.731_{\pm.020}$	$.592_{\pm.102}$	$.541_{\pm.031}$
	OC-SVM	Tabular	$.293_{\pm.034}$	$.747_{\pm.018}$	$.600_{\pm.024}$	$.570_{\pm.019}$
	DOMINANT	Graph	$.269_{\pm.030}$	$.739_{\pm.011}$	$.702_{\pm.010}$	$.555_{\pm.012}$
	TAM	Graph	$.130_{\pm.041}$	$.688_{\pm.032}$	$.423_{\pm.011}$	$.470_{\pm.029}$
	BlindGuard	MAS-specific	$.126_{\pm.029}$	$.687_{\pm.027}$	$.415_{\pm.015}$	$.467_{\pm.021}$
Semi-Supervised	DeepSAD	Tabular	$.490_{\pm.057}$	$.818_{\pm.009}$	$.803_{\pm.016}$	$.690_{\pm.030}$
	DevNet	Tabular	$.481_{\pm.062}$	$.815_{\pm.015}$	$.788_{\pm.015}$	$.684_{\pm.034}$
	GGAD	Graph	$.172_{\pm.053}$	$.704_{\pm.018}$	$.516_{\pm.027}$	$.496_{\pm.028}$
	G-Safe (semi-sup)	MAS-specific	$.324_{\pm.080}$	$.841_{\pm.013}$	$.797_{\pm.018}$	$.600_{\pm.038}$
	Gemma4-E2B (few-shot)	LLM	$.303_{\pm.019}$	$.739_{\pm.025}$	$.571_{\pm.027}$	$.572_{\pm.015}$
	Gemma4-E4B (few-shot)	LLM	$.342_{\pm.039}$	$.753_{\pm.033}$	$.591_{\pm.043}$	$.596_{\pm.030}$
	Gemma4-26B-A4B (few-shot)	LLM	$.312_{\pm.041}$	$.745_{\pm.031}$	$.613_{\pm.020}$	$.578_{\pm.029}$
	Gemma4-31B (few-shot)	LLM	$.348_{\pm.012}$	$.759_{\pm.026}$	$.628_{\pm.038}$	$.600_{\pm.016}$
Supervised	SVM	Tabular	$.553_{\pm.033}$	$.841_{\pm.006}$	$.796_{\pm.017}$	$.728_{\pm.016}$
	Random Forest	Tabular	$\mathbf{.594_{\pm.044}}$	$.855_{\pm.007}$	$.863_{\pm.007}$	$\mathbf{.754_{\pm.023}}$
	XGBoost	Tabular	$.587_{\pm.039}$	$.853_{\pm.006}$	$\mathbf{.878_{\pm.012}}$	$.749_{\pm.020}$
	G-Safeguard	MAS-specific	$.551_{\pm.030}$	$\mathbf{.855_{\pm.012}}$	$.834_{\pm.013}$	$.716_{\pm.012}$

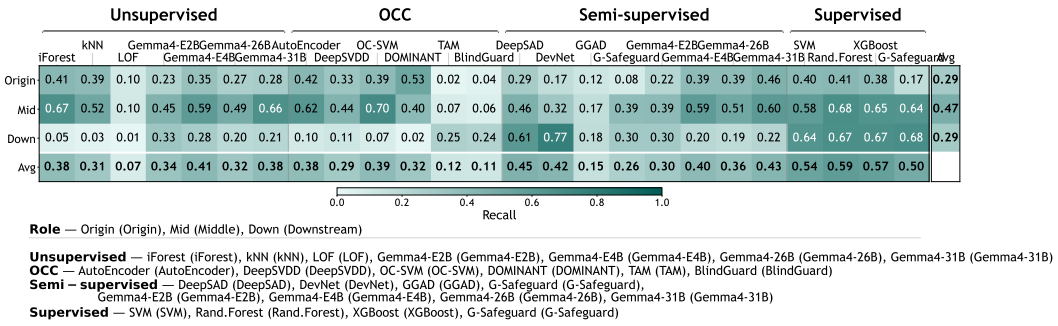


Figure 11: Per-method recall on multi-anomaly propagation chains, broken down by step role. **Rows**: step role, including **Origin** (the first anomalous step), **Mid** (intermediate anomalous steps), and **Down** (the last anomalous step); **Bottom row**: per-method average across roles, indicating overall propagation detection performance. **Columns**: AD methods grouped by supervision level. **Rightmost column**: per-role average across methods. Middle steps have the highest average recall across methods (0.47), while origin and downstream steps are substantially harder (0.29 and 0.29, respectively). Overall, Random Forest achieves the best average propagation-role recall (0.59), followed by XGBoost (0.57) and SVM (0.54). Per role, DOMINANT performs best on origin steps (recall 0.53), OC-SVM performs best on middle steps (recall 0.70), and DevNet performs best on downstream steps (recall 0.77).

Table 10: One-tailed two-sample t-test p-values (Method > Random) for overall anomaly detection performance across four performance metrics.

Supervision	Method	Method Family	F1	Acc	AUC	Bal. Acc
-	Random	Random	-	-	-	-
Unsupervised	Isolation Forest	Tabular	.001	.006	.022	$< .001$
	KNN	Tabular	.004	.081	.631	.006
	LOF	Tabular	1.000	.988	1.000	1.000
	Gemma4-E2B (zero-shot)	LLM	$< .001$	.005	$< .001$	$< .001$
	Gemma4-E4B (zero-shot)	LLM	$< .001$	.002	$< .001$	$< .001$
	Gemma4-26B-A4B (zero-shot)	LLM	.004	.039	.002	.002
	Gemma4-31B (zero-shot)	LLM	$< .001$	.009	$< .001$	$< .001$
One-Class	Autoencoder	Tabular	$< .001$	.006	.144	$< .001$
	DeepSVDD	Tabular	.024	.045	.049	.017
	OC-SVM	Tabular	$< .001$	.005	$< .001$	$< .001$
	DOMINANT	Graph	$< .001$	.010	$< .001$	$< .001$
	TAM	Graph	.968	.822	1.000	.952
	BlindGuard	MAS-specific	.992	.861	1.000	.986
Semi-Supervised	DeepSAD	Tabular	$< .001$	$< .001$	$< .001$	$< .001$
	DevNet	Tabular	$< .001$	$< .001$	$< .001$	$< .001$
	GGAD	Graph	.562	.528	.079	.564
	G-Safe (semi-sup)	MAS-specific	.006	$< .001$	$< .001$	.001
	Gemma4-E2B (few-shot)	LLM	$< .001$	.064	.014	.001
	Gemma4-E4B (few-shot)	LLM	.005	.055	.027	.011
	Gemma4-26B-A4B (few-shot)	LLM	.009	.066	.001	.015
	Gemma4-31B (few-shot)	LLM	$< .001$	.021	.010	$< .001$
Supervised	SVM	Tabular	$< .001$	$< .001$	$< .001$	$< .001$
	Random Forest	Tabular	$< \mathbf{.001}$	$< .001$	$< .001$	$< \mathbf{.001}$
	XGBoost	Tabular	$< .001$	$< .001$	$< \mathbf{.001}$	$< .001$
	G-Safeguard	MAS-specific	$< .001$	$< \mathbf{.001}$	$< .001$	$< .001$

Table 11: AUCROC per (method × action), mean ± std over available train-test splits. Action success rate shown in parentheses. Per-column maxima in **bold**

Family	Method	SP (87.0%)	OC (96.5%)	SC (76.0%)	OI (91.9%)	TC (62.5%)	AD (57.3%)	OP (99.5%)	Avg
Unsupervised	iForest	0.49 ± 0.13	0.54 ± 0.02	0.60 ± 0.06	0.56 ± 0.04	0.51 ± 0.04	0.53 ± 0.07	0.47 ± 0.24	0.53 ± 0.05
	kNN	0.37 ± 0.17	0.52 ± 0.07	0.68 ± 0.09	0.58 ± 0.04	0.54 ± 0.09	0.61 ± 0.03	0.52 ± 0.18	0.55 ± 0.10
	LOF	0.71 ± 0.16	0.49 ± 0.05	0.62 ± 0.07	0.52 ± 0.05	0.57 ± 0.08	0.59 ± 0.03	0.56 ± 0.23	0.58 ± 0.07
	Gemma4-E2B	0.24 ± 0.10	0.29 ± 0.08	0.62 ± 0.02	0.71 ± 0.05	0.40 ± 0.07	0.61 ± 0.02	0.41 ± 0.27	0.47 ± 0.18
	Gemma4-E4B	0.19 ± 0.03	0.46 ± 0.06	0.63 ± 0.02	0.65 ± 0.03	0.53 ± 0.05	0.54 ± 0.04	0.72 ± 0.11	0.53 ± 0.17
	Gemma4-26B	0.23 ± 0.04	0.51 ± 0.09	0.64 ± 0.02	0.56 ± 0.03	0.67 ± 0.03	0.59 ± 0.04	0.55 ± 0.24	0.54 ± 0.14
	Gemma4-31B	0.36 ± 0.06	0.65 ± 0.09	$\mathbf{0.77 \pm 0.02}$	0.65 ± 0.02	0.68 ± 0.04	0.65 ± 0.02	0.74 ± 0.05	0.64 ± 0.13
OCC	AutoEncoder	0.59 ± 0.15	0.54 ± 0.05	0.68 ± 0.08	0.58 ± 0.04	0.62 ± 0.09	0.60 ± 0.06	0.53 ± 0.21	0.59 ± 0.05
	DeepSVDD	0.48 ± 0.18	0.54 ± 0.05	0.62 ± 0.04	0.51 ± 0.05	0.50 ± 0.11	0.49 ± 0.03	0.59 ± 0.19	0.53 ± 0.05
	OC-SVM	0.72 ± 0.10	0.57 ± 0.04	0.66 ± 0.09	0.58 ± 0.04	0.59 ± 0.10	0.55 ± 0.04	0.58 ± 0.18	0.61 ± 0.06
	DOMINANT	0.26 ± 0.12	0.51 ± 0.04	0.59 ± 0.06	0.57 ± 0.05	0.37 ± 0.03	0.33 ± 0.05	0.50 ± 0.23	0.45 ± 0.13
	TAM	0.14 ± 0.06	0.29 ± 0.08	0.46 ± 0.09	0.50 ± 0.07	0.46 ± 0.08	0.54 ± 0.04	0.72 ± 0.15	0.44 ± 0.19
	BlindGuard	0.36 ± 0.06	0.48 ± 0.09	0.37 ± 0.09	0.51 ± 0.04	0.46 ± 0.07	0.49 ± 0.04	0.62 ± 0.26	0.47 ± 0.09
Semi-supervised	DeepSAD	0.85 ± 0.11	0.67 ± 0.06	0.68 ± 0.07	0.64 ± 0.03	0.64 ± 0.04	0.76 ± 0.03	0.56 ± 0.31	0.69 ± 0.09
	DevNet	0.80 ± 0.10	0.57 ± 0.05	0.64 ± 0.08	0.60 ± 0.07	0.62 ± 0.03	0.69 ± 0.07	0.60 ± 0.13	0.65 ± 0.08
	GGAD	0.51 ± 0.16	0.49 ± 0.08	0.53 ± 0.02	0.49 ± 0.05	0.52 ± 0.09	0.54 ± 0.08	0.48 ± 0.34	0.51 ± 0.02
	G-Safeguard	0.62 ± 0.09	0.63 ± 0.08	0.63 ± 0.03	0.70 ± 0.03	0.73 ± 0.02	0.74 ± 0.02	0.65 ± 0.24	0.67 ± 0.05
	Gemma4-E2B	0.44 ± 0.04	0.34 ± 0.08	0.62 ± 0.05	0.64 ± 0.05	0.40 ± 0.11	0.60 ± 0.03	0.51 ± 0.28	0.51 ± 0.12
	Gemma4-E4B	0.32 ± 0.05	0.54 ± 0.09	0.66 ± 0.05	0.61 ± 0.04	0.57 ± 0.10	0.58 ± 0.07	0.76 ± 0.26	0.58 ± 0.13
	Gemma4-26B	0.36 ± 0.05	0.67 ± 0.02	0.65 ± 0.03	0.66 ± 0.06	0.69 ± 0.04	0.60 ± 0.01	0.77 ± 0.18	0.63 ± 0.13
	Gemma4-31B	0.42 ± 0.08	0.85 ± 0.03	0.74 ± 0.05	0.78 ± 0.01	0.65 ± 0.10	0.65 ± 0.03	$\mathbf{0.89 \pm 0.08}$	0.71 ± 0.16
Supervised	SVM	0.82 ± 0.11	0.63 ± 0.06	0.73 ± 0.08	0.56 ± 0.04	0.73 ± 0.05	0.76 ± 0.03	0.68 ± 0.14	0.70 ± 0.09
	Rand.Forest	0.84 ± 0.11	0.78 ± 0.07	0.74 ± 0.06	0.73 ± 0.03	0.70 ± 0.07	0.79 ± 0.02	0.58 ± 0.30	0.74 ± 0.08
	XGBoost	$\mathbf{0.87 \pm 0.08}$	$\mathbf{0.89 \pm 0.04}$	0.72 ± 0.06	$\mathbf{0.81 \pm 0.03}$	0.72 ± 0.05	0.79 ± 0.02	0.65 ± 0.33	$\mathbf{0.78 \pm 0.09}$
	G-Safeguard	0.71 ± 0.06	0.65 ± 0.07	0.61 ± 0.06	0.73 ± 0.02	$\mathbf{0.78 \pm 0.03}$	$\mathbf{0.79 \pm 0.03}$	0.71 ± 0.15	0.71 ± 0.07
Avg		0.51 ± 0.23	0.56 ± 0.15	0.64 ± 0.09	0.62 ± 0.09	0.59 ± 0.11	0.62 ± 0.11	0.61 ± 0.11	0.59 ± 0.04