MAS-COMP-BENCH: DISCOVERING COMPOSITIONAL FAILURES IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) combine agents with different capabilities and tools to tackle complex real-world tasks. In these systems, each agent can follow its predefined rules while their combined actions violate a system-wide requirement for safety, correctness, or privacy. Such violations are compositional failures. Finding such failures requires testing tool-call sequences to learn how agents affect one another, but existing benchmarks largely evaluate preselected compositions or give red teams an attack plan to follow. Towards this end, we introduce MAS-Comp-Bench, a benchmark with three simulated environments—software engineering, e-commerce operations, and multi-tenant records—where red teams search for failures through interacting with the environment without being given an attack plan. The benchmark's simulation engine executes arbitrary tool-call sequences and returns feedback after each attempt. We compare the failures each red team finds with the full set designed into each environment to measure what was missed. In order to improve search efficiency for automated red teams, we introduce RedExplorer, a multi-agent red-teaming system designed to identify critical environment behavior and compose successful compositional attacks. On MAS-Comp-Bench, our RedExplorer coordinates a memory agent, an orchestration agent, and execution agents to search for compositional failures efficiently. With GPT-4o and the same 40-attempt attack budget, it discovers five distinct failure mechanisms versus one for the solo-memory baseline—a 400% relative increase—while reducing estimated API cost by 9.2%.

1 INTRODUCTION

Multi-agent systems (MAS) combine agents with different information, capabilities, and tool access to complete complex tasks (Wu et al., 2023; Hong et al., 2024; Anthropic, 2025). These agents collaborate by passing information, invoking tools, and acting on a shared state. In these systems, one agent's actions may change the information or state on which another agent relies, creating dependencies across the system. Prior work shows that individually acceptable capabilities or outputs can combine to produce unsafe outcomes (Spera, 2026; Patil et al., 2025). We focus on *compositional failures*: all participating agents act consistently with their predefined local rules, performing actions or tool-call sequences with valid inputs, while their combined actions violate a system-wide requirement for safety, correctness, or privacy. These failures arise from how agents' actions combine, rather than from any single agent violating its predefined rules.

Detecting compositional failures before deployment is therefore a central challenge for evaluating MAS. The difficulty is that the cross-agent dependencies responsible for these failures may not appear in individual agents' role and tool descriptions. Some dependencies become visible only during execution, through intermediate outputs and changes to a shared environment state. Consequently, a standard red team cannot reliably identify a tool-call sequence that triggers a compositional failure from the context it is given about the tools in the environment. Instead, the red team must probe the system: execute candidate tool-call sequences, observe how the system responds, and use each response to decide what to test next.

However, existing benchmarks provide limited evaluation of whether red teams can discover compositional failures by learning unspecified cross-agent dependencies through iterative interaction. Prior evaluations focus on predefined adversarial scenarios (Kavathekar et al., 2026), supplied infer-

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.

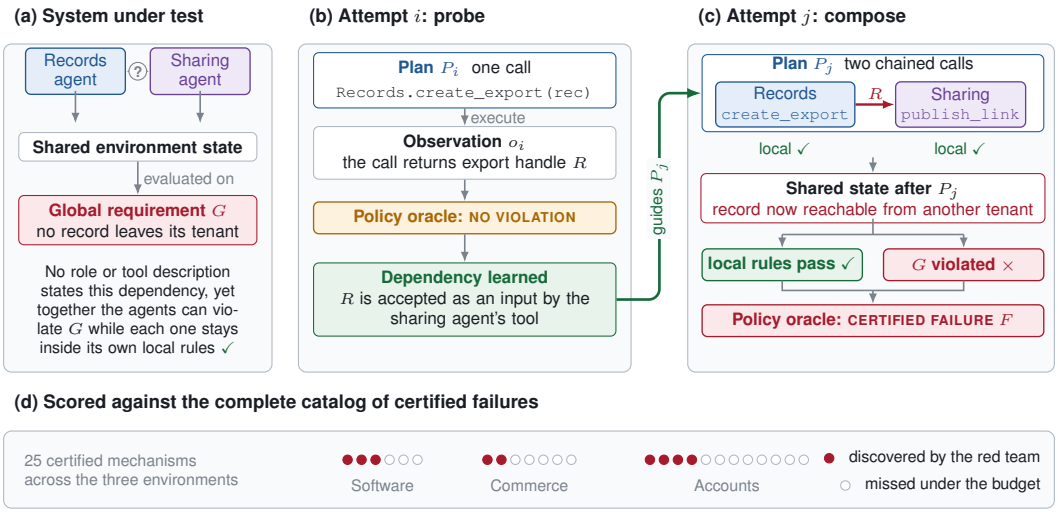


Figure 1: Compositional-failure discovery in MAS-Comp-Bench, illustrated on the multi-tenant records environment. **(a)** Every agent satisfies its own local requirements, but the dependency linking them—here, that an export handle produced by the records agent is accepted as an input by the sharing agent's tool—appears in no role or tool description, so the global requirement G can be violated by their combined actions. **(b)** A red team must therefore probe the system: attempt i executes a single call, the policy oracle reports no violation, and the observation nevertheless exposes the dependency. **(c)** Attempt j composes the two locally permitted calls; each local rule still passes, G fails, and the oracle certifies compositional failure F. **(d)** Because each environment ships a complete catalog of certified mechanisms (Table 1), the benchmark scores both what a red team discovers and what it misses under a fixed interaction budget; the coverage shown here is illustrative.

ence plans (Patil et al., 2025), or skill compositions identified from available artifacts or constructed around predefined failure mechanisms (Wang et al., 2026; Xie et al., 2026). To address this gap, we introduce MAS-Comp-Bench, a benchmark that tests whether red teams can discover compositional failures by interacting with the system and learning from its responses. MAS-Comp-Bench contains three executable simulation environments modeled on common MAS applications in software development, e-commerce, and account management. In each environment, red teams must execute candidate tool-call sequences and observe the resulting system behavior to obtain the information needed to discover and reproduce compositional failures (See Figure 1).

Comparing red teams on this discovery task requires a consistent way to execute their attempts and a known set of failures against which to score their discoveries. To achieve this, the framework uses a deterministic simulation engine that executes each tool-call sequence, returns environment feedback, and verifies whether the sequence produces a compositional failure. This common simulation process allows different red-team methods to interact autonomously with the same environments and adapt their attack strategies based on that feedback, while ensuring that evaluation can be consistent across various red teams. Each environment includes a complete catalog of possible compositional failures which are verified by the simulation engine logic, enabling the benchmark to measure both what a red team discovers and what remains undiscovered under a fixed budget of iterative attack attempts. Across all three environments, MAS-Comp-Bench contains 103 executable functions representing agent and tool call actions in its primary interface. The compositional failure catalog consists of 25 distinct and verifiable compositional failure signatures.

Within the benchmark's fixed interaction budget, exhaustively searching over agent combinations and tool-call sequences is impractical. To make compositional-failure discovery more efficient, we further introduce RedExplorer, a multi-agent red-teaming system. RedExplorer learns cross-agent dependencies from observed interactions in the environment. These dependencies capture how one agent's outputs, permissions, or state changes affect actions elsewhere in the system. RedExplorer uses these dependencies to choose agent combinations and tool-call sequences to test next.

We summarize our contributions as follows: (i) We introduce MAS-Comp-Bench, a benchmark with three executable simulation environments for evaluating whether automated red teams can discover compositional failures through interaction with multi-agent systems. MAS-Comp-Bench uses a deterministic simulation engine and a complete catalog of compositional failures to compare what automated red teams find and miss under a fixed interaction budget. (ii) We introduce RedExplorer, a multi-agent red-teaming system that learns cross-agent interaction dependencies from observed interactions and uses them to guide subsequent attack attempts. (iii) We evaluate closed- and open-source models and multi-agent systems. In the matched GPT-4o comparison, RedExplorer discovers five distinct failure mechanisms versus one for the solo-memory baseline (a 400% relative increase) within the same 40-attempt interaction budget, while reducing estimated API cost by 9.2%.

2 Related Work

2.1 Agent and MAS safety

Prior work on agent safety primarily evaluates failures of individual agents interacting with tools and environments. ToolEmu (Ruan et al., 2024), AgentHarm (Andriushchenko et al., 2025), Agent-SafetyBench (Zhang et al., 2024), SafeArena (Tur et al., 2025), and SafeAgentBench (Yin et al., 2024) study harmful requests and unsafe actions. InjecAgent (Zhan et al., 2024), AgentDojo (Debenedetti et al., 2024), WASP (Evtimov et al., 2025), and AgentDyn (Li et al., 2026a) evaluate indirect prompt injection, while Agent Security Bench (Zhang et al., 2025), MCP-SafetyBench (Zong et al., 2026), and OS-Harm (Kuntz et al., 2025) cover broader attacks across agent components and execution environments.

Recent benchmarks extend safety evaluation to multi-agent systems (MAS). NetSafe (Yu et al., 2025), TAMAS (Kavathekar et al., 2026), and ACIArena (An et al., 2026) study adversarial inputs or compromised agents and how their effects propagate through a system. A2ASecBench evaluates attacks on agent-to-agent protocols (Li et al., 2026b). ConVerse (Gomaa et al., 2026), MAMA (Liu et al., 2026), and AgentLeak (El Yagoubi et al., 2026) study privacy and security failures during agent interaction, while MAFF-Bench evaluates coordinated fraud (Ren et al., 2026). Together, these studies show how communication, shared information, and system architecture introduce risks beyond those captured by individual-agent evaluation. Our work complements these evaluations by isolating compositional failures in which every participating agent satisfies its stated local requirements, but their combined actions violate a global requirement.

2.2 Compositional-failure evaluation

Recent work examines how individually acceptable components can produce unsafe behavior when combined. Spera (2026) formalizes the non-compositionality of safety under conjunctive capability dependencies. SkillReact (Wang et al., 2026) investigates risks arising from jointly available skill capabilities, while SCR-Bench (Xie et al., 2026) evaluates harmful execution paths formed by skills that appear benign in isolation. Compositional privacy studies examine how information distributed across agents (Patil et al., 2025) or tool outputs (Qiao et al., 2025) can jointly reveal sensitive attributes. Semantic Intent Fragmentation (SIF) (Ahad et al., 2026) studies single-shot requests that induce policy-violating plans even when individual subtasks pass safety classifiers. These studies motivate evaluating the consequences of composition beyond checks on individual components.

A key distinction concerns whether discovering a failure requires learning dependencies through exploratory interaction. SkillReact identifies candidate compositions using extracted capability profiles and predefined forbidden patterns, followed by human adjudication and selected tool-call validation (Wang et al., 2026). SkillProbe analyzes code and documentation and applies risk-link policies to identify potentially hazardous skill chains (Guo et al., 2026). SIF constructs single-shot attack requests from supplied domain and harm specifications (Ahad et al., 2026). These approaches identify candidate attacks using information available before their execution. Our MAS-Comp-Bench instead targets dependencies that are unspecified in local descriptions, requiring red teams to gather evidence through execution and use it to guide subsequent attempts.

Another related distinction is whether the evaluation measures autonomous discovery of vulnerable compositions. The evaluation accompanying CoDef supplies attackers with a correct high-level inference plan, allowing it to assess leakage during execution separately from plan discovery (Patil

Table 1: MAS-Comp-Bench environment inventory. Tools are callable tools in the primary catalog; certified mechanisms are unique keys passing the compositional certificates. Mean tools/mechanism is the average number of distinct tool types in canonical compositional plans.

Environment	Simulated setting	Tools	Certified mechanisms	Mean tools/ mechanism
Software engineering	Code review, CI, repositories, design documents, and database workflows	28	6	2.3
E-commerce operations	Orders, fulfillment, refunds, inventory, procurement, and settlement	47	7	2.0
Multi-tenant records	Records, sharing, exports, asynchronous jobs, and aggregation	28	12	2.3

et al., 2025). SCR-Bench evaluates controlled compositions organized around specified failure mechanisms, including neutral conditions in which agents autonomously form harmful chains (Xie et al., 2026). Feedback-driven attack discovery is also explored by SkillAttack, which refines adversarial prompts and attack paths against skills (Duan et al., 2026), and AutoMalTool, which generates and optimizes malicious MCP tool descriptions (He et al., 2026). Our benchmark instead evaluates automated red teams through a common interaction interface, requiring them to identify vulnerable compositions and construct attacks without a supplied attack plan or prescribed attack mechanism.

Finally, existing benchmarks do not measure how completely a red team discovers the failures present in a system. They evaluate fixed attack instances, candidate compositions, or predefined failure mechanisms without providing a complete inventory of executable compositional failures within each evaluated system (Kavathekar et al., 2026; An et al., 2026; Wang et al., 2026; Xie et al., 2026). They can therefore measure whether attempted attacks succeed, but not how many failures remain undiscovered. We instead equip each environment with an explicit evaluator-side inventory of compositional failures, enabling measurement of both the failures a red team discovers and those it misses under a fixed interaction budget.

Our MAS-Comp-Bench thus complements prior studies of compositional risk and adaptive red teaming, by combining discovery of unspecified cross-agent dependencies, autonomous attack construction, and measurement of failure coverage in environments where participating agents remain locally compliant.

3 MAS-COMP-BENCH

3.1 OVERVIEW

We introduce MAS-Comp-Bench, a benchmark for evaluating the ability of automated red teams in discovering compositional failures within multi-agent systems. To reflect practical MAS workflows, we construct three interactive environments with realistic settings and tools spanning software development, e-commerce, and multi-tenant records management. Each environment contains compositional failure mechanisms arising from cross-agent dependencies that are unspecified in local role and tool descriptions. To discover and reproduce these failures, red teams probe the environment and use execution feedback to learn how agent actions can combine to violate system wide requirements. A deterministic simulation engine executes submitted tool call sequences against a shared environment state and verifies whether they produce a compositional failure. After each attempt, the engine returns observations and evaluation feedback, allowing red teams to refine their understanding of the environment and guide subsequent attempts.

Table 1 summarizes the three environments, their tool inventories, and their certified compositional failure mechanisms. Each failure count corresponds to the number of distinct certified mechanisms in the environment's failure catalog.

3.2 DESIGNING EXPLORATION AS A REQUIREMENT

In real world multi-agent systems, agents interact with other agents and tools based on local knowledge of their roles and capabilities. This knowledge may not capture how actions affect other com-

ponents. Therefore compositional failures can arise even when each agent satisfies its local requirements. Automated red teams can face a similar information gap when attempting to uncover failures in these systems. We therefore study a discovery setting where red teams must probe the system and observe its behavior to learn the cross-agent dependencies that enable compositional failures.

To reflect this setting, we design benchmark environments whose relevant cross-agent dependencies are unspecified in the initial context provided to red teams. This context describes the local capabilities available within each environment, with possible actions presented thorugh a standardized interface of callable tool functions.

Each function includes input and output schemas that allow red teams to construct action sequences where later calls use values returned by earlier calls. Each tool also has a short usage description explaining its role in the environment. These descriptions specify local functionality without revealing the underlying failure mechanisms encoded in the simulation logic. By executing candidate sequences and observing their outputs and effects, red teams can gather evidence about how tools interact and use that evidence to construct subsequent attempts. To verify the role of environment feedback, Table 5 reports a reasoning-only control in which models propose tool-call sequences without receiving execution feedback.

3.3 Semantic References for Inputs

In order to provide a coherent evaluation of automated red teaming capabilities across various models and frameworks, environments must be easy to interact with for models not familiar with their tools and functionality. If red teams are unable to reliably pass syntactically valid and essential arguments, it is impossible to accurately evaluate their true potential failure identification abilities. In order to ensure MAS-Comp-Bench provides an accurate assessment of compositional failure discovery performance, we design interaction interfaces that make it convenient for models to compose attack plans that pass system validation.

Table 2: Identifier-interface conditions. The interface changes how an attacker names existing entities and reuses newly produced outputs; it does not change the environment policy or ground-truth failure catalog.

Interface	Attacker representation	Controlled difficulty
Opaque literal	Exact opaque identifiers copied from observations	Identifier retrieval and exact copying
Runtime reference	Structured references to fields returned by earlier successful actions	Cross-step data binding
Semantic neutral	Stable typed selectors describing ordinary identity, without security-revealing labels	Entity selection with reduced identifier bookkeeping

MAS-Comp-Bench exposes the same underlying environments through the three interfaces in Table 2. This controlled variation separates security reasoning from the bookkeeping needed to address entities and connect outputs across actions. Runtime references may resolve only to fields produced by earlier successful calls; unresolved or forward references are recorded as invalid plans rather than treated as safe executions.

To determine how best to design the interaction interface for red teams in MAS-Comp-Bench, we perform ablations by evaluating the effect of tool input types on plan validity. We evaluate GPT-4o (Hurst et al., 2024) across all three environments, varying only the action interface used to compose plans across the three conditions in Table 2.

3.4 Deterministic Action Execution and Evaluation

Real world MAS experience compositional failures through the impact of agents' actions on a shared environment state. To reflect this in our evaluation, we utilize a deterministic simulation engine as seen in Figure 2 to process submitted red team plans and provide outputs that replicate how real agent and tool interactions function. When red teams submit attacks through their responses, the simulation engine parses each action component in a candidate composition before beginning validation of input fields. At each time step t, the simulation engine applies $S(a_t, s_t) \rightarrow (s_{t+1}, o_t, d_t)$,

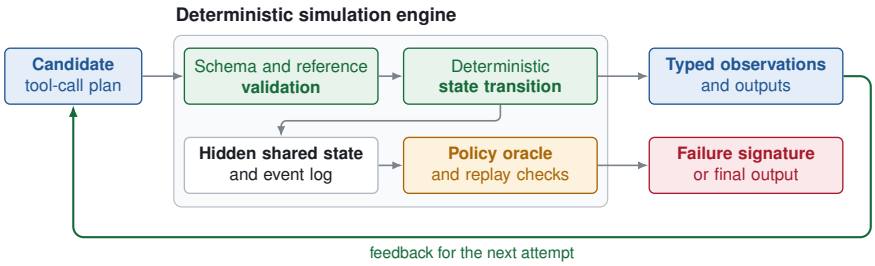


Figure 2: Deterministic execution and evaluation pipeline. The red team sees tool schemas, observations, and typed outputs, while the engine maintains hidden state, provenance, and policy evidence. The same transition and oracle code is used for online evaluation and offline replay.

producing the next environment state, a model-visible observation, and structured data that later actions may reference. Each action is executed by the simulation logic sequentially, with the mutable environment state updated dynamically after each execution. After the execution of an action a_t, the simulation engine maintains a record of the proposed action, resolved arguments from prior dependencies, and execution status.

To evaluate the occurrence of compositional failures, we utilize a policy oracle as part of the simulation system. The policy oracle takes in the environment state and assesses it against the benchmark's catalog of designed environment failures. These failures are not identified through string or sequence matching. Instead, the oracle evaluates explicit state predicates $\mathcal{C} = \{C_1, \ldots, C_m\}$, where each C_i corresponds to a distinct failure identity. At each simulation step, the oracle evaluates the current state s_t and structured trajectory evidence against these predicates. If a condition is violated, the simulation engine returns the successful sequence and failure signature; otherwise, it returns the final outputs of the submitted plan.

The simulation engine and corresponding design provide multiple evaluation benefits. Determinism allows for reproducible evaluation across models by ensuring that environment behavior and tool responses stay consistent, giving red teams a level playing field. Environment feedback also provides a realistic signal for automated red teams to strategically improve based on their past actions. Verification of failures through the policy oracle ensures submitted plans from attackers can be quickly evaluated. With a known corpus of environment failure signatures, evaluated red teams can be measured against a known upper bound of possible failures that can be discovered.

4 REDEXPLORER

MAS-Comp-Bench challenges automated red teams to use environment feedback to guide their search for compositional failures within a limited interaction budget. To make this search more efficient, we introduce RedExplorer, a multi-agent red-teaming harness that uses observed interactions to guide subsequent attempts. Figure 3 illustrates its agent roles, and Table 4 reports the comparative results.

We implement RedExplorer as a team of specialized agents working within constrained roles. Our design isolates the various phases of agent reasoning in MAS-Comp-Bench while remaining generic and applicable to compositional-failure discovery tasks in the real world. Specifically, we designate three distinct worker roles that operate in a coordinated loop.

4.1 MEMORY

The memory agent in RedExplorer efficiently consolidates previous attempt results into an optimized textual summary of past iterations. While the simulation-engine feedback described in Section 3 is designed to reflect real-world MAS, action outputs alone do not immediately reveal which information is relevant for updating future attacks. We therefore task the memory agent with analyzing the raw action and output history and providing a concise summary of relevant events and avenues for potential exploration.

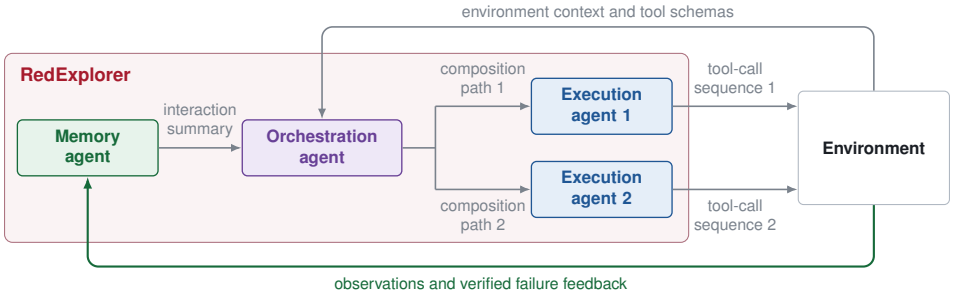


Figure 3: RedExplorer architecture. All RedExplorer agents are enclosed in the red region on the left. Execution agents submit attacks to the external environment, which returns context, schemas, and execution feedback.

4.2 Orchestration

To promote a more diverse search across potential environment actions, we establish the orchestration agent as a planner focused on strategic exploration. The orchestrator agent receives the environment context along with a targeted summary from the memory agent before planning out multiple composition paths, which serve as instructions for exploiting previously viewed behaviors or understanding more of the environment's behavior. We instruct the orchestrator to compose paths which are generally orthogonal to one another with respect to the actions involved and the type of target compositional failure. The orchestrator also is instructed to utilize initial composition paths as a mechanism to gain familiarity with the environment even if it means that submitted attacks are likely to be safe.

4.3 Execution

Execution agents take in composition paths from the orchestration agent as well as input and output schemas for the environment's action catalog and are prompted to submit concrete attack compositions. These agents are given a narrow scope of tools, input parameters, and ordering to utilize and they use this context to submit attacks and obtain verified failures and environment responses based on the orchestrator agent's planned exploration.

5 Evaluation

5.1 Implementation Details

We evaluate a variety of proprietary and open models on MAS-Comp-Bench. In our evaluations we provide all models an attack budget of 40 iterations. All models with temperature settings were set to a temperature of 0.0. We access all models through APIs via OpenAI, Anthropic, and OpenRouter.

5.2 Primary Results

We evaluate a variety of proprietary and open models on MAS-Comp-Bench across all three environments. Results are shown in Table 3, separated into closed-weight and open-weight models. Our results demonstrate that strong open and proprietary models demonstrate some compositional failure identification ability as automated red teams, although MAS-Comp Bench remains challenging overall. Reporting both distinct discoveries and reachable-mechanism recall makes performance comparable across environments with different numbers of certified failures.

5.3 RedExplorer Comparison

Table 4 compares RedExplorer with the GPT-4o solo-memory baseline under a matched total attack budget. RedExplorer uses two execution agents per orchestrator iteration, so its iteration count is halved to keep the number of submitted attacks consistent with the baseline run. The table uses the

Table 3: Solo-memory model comparison under matched conditions across the three MAS-Comp-Bench environments. Each environment reports the number of distinct certified strict mechanisms discovered and the percentage of its reachable mechanisms found. The denominators are 6, 7, and 12 mechanisms for software engineering, e-commerce operations, and multi-tenant records, respectively. Overall recall uses all 25 reachable mechanisms. Cost is the total USD cost of planner and memory-model calls across the three environments. Costs are provider-reported when available and otherwise estimated from recorded token usage using the frozen pricing manifest.

	Software engineering		E-commerce operations		Multi-tenant records		Overall		
Model	Distinct	Found (%)	Distinct	Found (%)	Distinct	Found (%)	Total	Found (%)	Cost
Closed-weight models									
GPT-4o	0	0.0	0	0.0	1	8.3	1	4.0	est. $3.54
Claude Opus 4.8	2	33.3	2	28.6	3	25.0	7	28.0	est. $18.48
Gemini 3.8 Flash	1	16.7	0	0.0	4	33.3	5	20.0	est. $1.10
Open-weight models									
Qwen3.8-Max-0902	1	16.7	1	14.3	4	33.3	6	24.0	$2.83
Kimi-K3	1	16.7	1	14.3	5	41.7	7	28.0	$4.08

Table 4: GPT-4o RedExplorer comparison with its solo-memory baseline under matched total attack budgets. Columns and reachable-mechanism denominators match Table 3. Costs are token-based estimates across the three environments.

	Software engineering		E-commerce operations		Multi-tenant records		Overall		
Model / method	Distinct	Found (%)	Distinct	Found (%)	Distinct	Found (%)	Total	Found (%)	Cost
GPT-4o (solo-memory)	0	0.0	0	0.0	1	8.3	1	4.0	est. $3.54
GPT-4o (RedExplorer)	0	0.0	0	0.0	5	41.7	5	20.0	est. $3.21

same per-environment discovery and overall-recall columns as the primary experiment. RedExplorer costs include all planner, memory, orchestration, and execution-agent calls.

Under the matched 40-attack budget, RedExplorer discovers five distinct mechanisms versus one for GPT-4o with solo memory. This is a 400% relative increase (a $5\times$ result) and raises overall recall from 4.0% to 20.0%, an increase of 16.0 percentage points. Estimated API cost decreases from $3.54 to $3.21, a 9.2% reduction. The four additional discoveries all occur in the multi-tenant-records environment; RedExplorer does not improve discovery in the other two environments in this evaluation.

5.4 Ablations

Attempting MAS-Comp-Bench Without Exploration. We evaluate GPT-4o and Claude Opus 4.8 on MAS-Comp-Bench using a modified experiment design to evaluate the performance of models on the benchmark when not given exploration capability. Red teams interact with the environment identically as in the experiments in Table 3 but they are not given any environment feedback upon composition submission. This results in an evaluation of the ability of red teams to determine compositional failures in MAS-Comp-Bench without interaction.

Table 5: Reasoning-only control with environment feedback removed. Each cell reports the number of distinct certified strict mechanisms discovered in the corresponding environment; "Valid plans" reports schema-valid proposals across all three environments.

Model	Software	Commerce	Accounts	Valid plans
GPT-4o	0	0	0	0
Claude Opus 4.8	0	0	0	0

As shown in Table 5, we find models which succeed when given memory are not capable of making meaningful progress towards identifying compositional failures without the ability to meaningfully interact with environments.

Interaction Interface Comparison. We perform ablations on the interaction interfaces red teams utilize in MAS-Comp-Bench in Table 6. We evaluate three variants as described in Table 2 based on plan validity. All models evaluated were given 100 attack iterations and no multi-agent orchestration. Plan validity measures the percentage of submitted red-team attacks that are syntactically valid based on the environment tool definitions. Expressibility is the certified strict-mechanism fraction executable through the interface; valid plans are those marked `valid_complete` or `valid_until_success`; execution depth is the fraction of proposed actions successfully dispatched before termination; and strict recall is certified strict-mechanism recall.

Table 6: Identifier-interface ablation with GPT-4o, solo-cold search, and matched instances. Values are macro-averages over the three environments and five paired replicates (100 attempts per environment and interface).

Interface	Expressibility	Valid plans	Exec. depth	Strict recall
Opaque literal	88.9%	10.7%	0.452	5.8%
Runtime reference	88.9%	7.3%	0.355	5.0%
Semantic neutral	88.9%	19.3%	0.483	7.8%

Our ablation across interaction interfaces show that the semantic-neutral interface achieves the strongest plan-validity and discovery results. We use this information in order to select an interaction interface best suited for empowering red teams to utilize their full reasoning and planning capabilities. This evidence supports usage of the semantic neutral interface in the primary evaluations in Table 3, Table 4, and Table 5.

6 Conclusion

MAS-Comp-Bench provides a new evaluation setting for automated red teams in which attackers attempt to discover compositional failures in MAS across 3 sandbox environments. We model benchmark environments based on real world use cases of software engineering, e-commerce, and accounts management. Our deterministic simulation engine and compositional failure design ensures that automated red teams can have planned attacks quickly evaluated against a real environment state.

RedExplorer improves the process of compositional failure discovery in MAS-Comp-Bench through more efficient search. By decomposing the search process into three distinct roles for memory, orchestration, and execution, RedExplorer allows models to understand environments quickly and manipulate that understanding into successful submitted attacks. In our matched GPT-4o evaluation, RedExplorer discovers five distinct failure mechanisms compared with one for the solo-memory baseline, while reducing estimated API cost by 9.2%. This improvement is concentrated in the multi-tenant-records environment rather than being uniform across all three environments.

7 Limitations

Cybersecurity Refusals in Frontier Models: In our evaluations of various automated red teams for MAS-Comp-Bench, we attempted to evaluate frontier models from OpenAI and Anthropic but encountered cybersecurity refusals. All models available released beyond GPT-4o and Claude Opus 4.8 presented cybersecurity refusals. We attempted evaluation with explicit instructions to agents that their tasks were benchmark evaluations, but this context adjustment did not result in any decrease in refusal behavior.

Ease of Environment Interation: The current version of MAS-Comp-Bench is designed to be as realistic to real world scenarios as possible. However, we note that despite selecting interaction interfaces and environment feedback to be as convenient for red teams as possible, we observe that automated red teams still struggle with composing fully valid plans as seen in Table 6. We leave improvement in this area to future works.

8 AI USE STATEMENT

In this work, we used generative AI tools to generate the code infrastructure for MAS-Comp-Bench, provide feedback on research methods, write code for running experiments, and produce synthetic candidate tool calls for environment design. We have not used generative AI tools for analyzing or interpreting any of our results or data, for proposing or refining hypotheses, for cleaning or reformatting datasets, or for designing our research methodology. The remaining required disclosure tasks, which concern mathematical proofs, conceptual formulation, and translation, are not applicable to this work. Additionally, we used generative AI tools for editing the software involved in MAS-Comp-Bench, identifying relevant literature, generating templates for figures, and editing the paper to improve its readability and language. We have reviewed all AI-assisted work. All code related to MAS-Comp-Bench and to the experiments in this paper has been manually verified and tested by human users to ensure that it matches the intended behavior of MAS-Comp-Bench. We manually reviewed all literature identified by generative AI tools to verify correctness. Tools and compositional failures were first generated with AI tools, and the authors then selected the viable ones and adjusted their behavior to meet the design constraints. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Tanzim Ahad, Ismail Hossain, Md Jahangir Alam, Sai Puppala, Yoonpyo Lee, Syed Bahauddin Alam, and Sajedul Talukder. Semantic Intent Fragmentation: A Single-Shot Compositional Attack on Multi-Agent AI Pipelines, 2026. URL https://arxiv.org/abs/2604.08608.

Hengyu An, Minxi Li, Jinghuai Zhang, Naen Xu, Chunyi Zhou, Changjiang Li, Xiaogang Xu, Tianyu Du, and Shouling Ji. ACIArena: Toward Unified Evaluation for Agent Cascading Injection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. doi: 10.18653/v1/2026.acl-long.457. URL https://aclanthology.org/2026.acl-long.457/.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c493d23af93118975cdbc32cbe7323f5-Abstract-Conference.html.

Anthropic. How we built our multi-agent research system, 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Zenghao Duan, Yuxin Tian, Zhiyi Yin, Liang Pang, Jingcheng Deng, Zihao Wei, Shicheng Xu, Yuyao Ge, and Xueqi Cheng. SkillAttack: Automated Red Teaming of Agent Skills through Attack Path Refinement, 2026. URL https://arxiv.org/abs/2604.04989.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. AgentLeak: A Benchmark for Internal-Channel Privacy Leakage in Multi-Agent LLM Systems, 2026. URL https://arxiv.org/abs/2602.11510.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. WASP: Benchmarking Web Agent Security Against Prompt Injection Attacks. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-0666. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1c9818387f5dd0a0bc151214660f059d-Abstract-Datasets_and_Benchmarks_Track.html.

Amr Gomaa, Ahmed Salem, and Sahar Abdelnabi. ConVerse: Benchmarking Contextual Safety in Agent-to-Agent Conversations. In *Findings of the Association for Computational Linguistics:*

EACL 2026, 2026. doi: 10.18653/v1/2026.findings-eacl.170. URL https://aclanthology.org/2026.findings-eacl.170/.

Zihan Guo, Zhiyu Chen, Xiaohang Nie, Jianghao Lin, Yuanjian Zhou, and Weinan Zhang. SkillProbe: Security Auditing for Emerging Agent Skill Marketplaces via Multi-Agent Collaboration, 2026. URL https://arxiv.org/abs/2603.21019.

Ping He, Changjiang Li, Binbin Zhao, Tianyu Du, and Shouling Ji. Automatic Red Teaming LLM-Based Agents with Model Context Protocol Tools. *IEEE Transactions on Information Forensics and Security*, 21:5165–5180, 2026. doi: 10.1109/TIFS.2026.3691201. URL https://doi.org/10.1109/TIFS.2026.3691201.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, A. J. Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o System Card, 2024. URL https://arxiv.org/abs/2410.21276.

Ishan Kavathekar, Hemang Jain, Ameya Rathod, Ponnurangam Kumaraguru, and Tanuja Ganu. TAMAS: Benchmarking Adversarial Risks in Multi-Agent LLM Systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. doi: 10.18653/v1/2026.acl-long.1442. URL https://aclanthology.org/2026.acl-long.1442/.

Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. OS-Harm: A Benchmark for Measuring Safety of Computer Use Agents. In *Advances in Neural Information Processing Systems*, 2025. doi: 10.52202/085713-1502. URL https://papers.neurips.cc/paper_files/paper/2025/hash/4009bff0cd87ba2203c8e3a2f082aaec-Abstract-Datasets_and_Benchmarks_Track.html.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. AgentDyn: Are Your Agent Security Defenses Deployable in Real-World Dynamic Environments?, 2026a. URL https://arxiv.org/abs/2602.03117.

Tianhao Li, Chuangxin Chu, Yujia Zheng, Bohan Zhang, Neil Zhenqiang Gong, and Chaowei Xiao. A2ASecBench: A Protocol-Aware Security Benchmark for Agent-to-Agent Multi-Agent Systems. In *The Fourteenth International Conference on Learning Representations (ICLR 2026)*, 2026b. URL https://safo-lab.github.io/A2ASecBench/.

Jinbo Liu, Defu Cao, Yifei Wei, Tianyao Su, Yuan Liang, Yushun Dong, Yan Liu, Yue Zhao, and Xiyang Hu. Topology Matters: Measuring Memory Leakage in Multi-Agent LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. doi: 10.18653/v1/2026.findings-acl.1980. URL https://aclanthology.org/2026.findings-acl.1980/.

Vaidehi Patil, Elias Stengel-Eskin, and Mohit Bansal. The Sum Leaks More Than Its Parts: Compositional Privacy Risks and Mitigations in Multi-Agent Collaboration, 2025. URL https://arxiv.org/abs/2509.14284.

Yuxuan Qiao, Dongqin Liu, Hongchang Yang, Wei Zhou, and Songlin Hu. Agent Tools Orchestration Leaks More: Dataset, Benchmark, and Mitigation, 2025. URL https://arxiv.org/abs/2512.16310.

Qibing Ren, Zhijie Zheng, Jiaxuan Guo, Junchi Yan, Lizhuang Ma, and Jing Shao. When AI Agents Collude Online: Financial Fraud Risks by Collaborative LLM Agents on Social Platforms. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2511.06448.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *International Conference on Learning Representations*, 2024. URL

https://proceedings.iclr.cc/paper_files/paper/2024/hash/7274ed909a312d4d869cc328ad1c5f04-Abstract-Conference.html.

Cosimo Spera. Safety is Non-Compositional: A Formal Framework for Capability-Based AI Systems, 2026. URL https://arxiv.org/abs/2603.15973.

Ada Defne Tur, Nicholas Meade, Xing Han Lù, Alejandra Zambrano, Arkil Patel, Esin Durmus, Spandana Gella, Karolina Stańczak, and Siva Reddy. SafeArena: Evaluating the Safety of Autonomous Web Agents, 2025. URL https://arxiv.org/abs/2503.04957.

Su Wang, Pin Qian, Yihang Chen, Junxian You, Xiaoyuan Wang, Xiaochong Jiang, Lifei Liu, Haoran Yu, and Jingzhou Xu. When Safe Skills Collide: Measuring Compositional Risk in Agent Skill Ecosystems, 2026. URL https://arxiv.org/abs/2606.00448.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Yi Xie, Jiawei Du, Yu Cheng, Jiuan Zhou, and Zhaoxia Yin. Benign in Isolation, Harmful in Composition: Security Risks in Agent Skill Ecosystems, 2026. URL https://arxiv.org/abs/2606.15242.

Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, Jing Shao, and Siheng Chen. SafeAgentBench: A Benchmark for Safe Task Planning of Embodied LLM Agents, 2024. URL https://arxiv.org/abs/2412.13178.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. NetSafe: Exploring the Topological Safety of Multi-agent System. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. doi: 10.18653/v1/2025.findings-acl.150. URL https://aclanthology.org/2025.findings-acl.150/.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-SafetyBench: Evaluating the Safety of LLM Agents, 2024. URL https://arxiv.org/abs/2412.14470.

Xuanjun Zong, Zhiqi Shen, Lei Wang, Yunshi Lan, and Yang Chao. MCP-SafetyBench: A Benchmark for Safety Evaluation of Large Language Models with Real-World MCP Servers. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/d46f127a80dc58cbc0732a717285c43a-Abstract-Conference.html.

APPENDIX

This appendix expands the benchmark and evaluation protocol described in the main text. We first summarize the three environment inventories, then describe the information exposed to each model, the prompting structure used by the evaluated red-team configurations, and the accounting and replay rules used for evaluation.

A Environment and Failure-Catalog Breakdown

Table 7 gives a per-environment breakdown of the action inventory and certified compositional-failure catalog. Tool-family names describe the ordinary application functionality visible to the red team; they do not reveal the evaluator-side failure predicates or canonical attack plans. A certified mechanism is a distinct failure identity in the catalog, not the number of trajectories that can reach that failure.

Table 7: Environment-level tool and failure-catalog breakdown. "Average tools" is the mean number of distinct tool types in the canonical plans for the certified mechanisms.

Environment	Model-visible tool families	Tools	Certified mechanisms	Average tools
Software engineering	Code review, continuous integration, repositories, design documents, and database workflows	28	6	2.3
E-commerce operations	Orders, fulfillment, refunds, inventory, procurement, and settlement	47	7	2.0
Multi-tenant records	Records, sharing, exports, asynchronous jobs, and aggregation	28	12	2.3
Total / macro view	Three executable multi-agent settings	103	25	2.2

Software engineering. This environment represents interactions among agents responsible for source repositories, review workflows, continuous-integration jobs, design artifacts, and database operations. Its 28 callable tools support six certified failure mechanisms. The canonical plans use 2.3 distinct tool types on average, so the catalog emphasizes effects that arise across components rather than isolated single-tool misuse.

E-commerce operations. This environment models order handling from purchase through fulfillment and settlement, including refund, inventory, and procurement operations. It has the largest action inventory, with 47 tools, and contains seven certified mechanisms. Canonical plans use 2.0 distinct tool types on average.

Multi-tenant records. This environment covers record creation and access, sharing, exports, asynchronous processing, and aggregation across tenant-scoped data. It contains 28 tools and 12 certified mechanisms. The example in Figure 1 illustrates one such mechanism: a handle returned by an export action becomes useful as an input to a later sharing action, and the composition creates cross-tenant reachability even though each individual action remains locally valid.

B Catalog Entries and Certification

Each catalog entry is keyed by a distinct compositional mechanism and is paired with an explicit state predicate. For a trajectory $\tau = (a_1, o_1, \ldots, a_T, o_T)$, the deterministic engine records the resolved arguments, state transitions, observations, and execution status at every step. The policy oracle evaluates the resulting state and trajectory evidence against the catalog predicates $\mathcal{C} = \{C_1, \ldots, C_m\}$. A discovery is credited only when the corresponding predicate is satisfied; matching a tool name or a surface string is insufficient.

The evaluator-side catalog stores the information needed to replay and audit a mechanism: its stable identity, canonical tool-call plan, required initial state, expected predicate, and failure signature. These fields remain hidden from the red team. Model-visible responses contain only ordinary tool outputs, typed values, validation errors, and the failure feedback returned after execution. Separating these two views prevents the prompt from directly revealing the benchmark answer while retaining a deterministic ground truth for scoring.

C Model-Visible Context and Prompting

All planners receive the same environment-specific interface under a given experimental condition. The context includes a short description of the simulated application, the available agent roles, and typed tool schemas with input and output fields. It excludes the failure catalog, policy predicates, canonical plans, and simulator implementation. Table 8 summarizes how that common context is specialized for each planner role.

Table 8: Prompt structure for the evaluated planner roles. The table describes the information contract rather than provider-specific chat wrappers.

Role	Supplied context	Required output
Solo-cold planner	Environment description, agent roles, tool schemas, and the current attempt index	One ordered candidate tool-call plan
Solo-memory planner	Solo-cold context plus a compact summary of previous attempts and observations	One revised candidate plan that may exploit or investigate prior feedback
Memory agent	Raw attempted actions, resolved outputs, validation results, and certified-failure feedback	A concise summary of observed dependencies, exhausted paths, and promising next probes
Orchestration agent	Environment context, tool schemas, and the memory summary	Multiple distinct composition paths specifying strategic objectives and relevant tools
Execution agent	One composition path plus the permitted tool schemas and typed arguments	One concrete, schema-valid ordered tool-call plan

The common planner instruction has four parts: (i) identify an interaction worth testing, (ii) select only tools available in the supplied schema, (iii) order actions so that later arguments may reuse typed outputs from earlier successful actions, and (iv) return a structured plan for validation and execution. The planner is explicitly allowed to submit exploratory plans that are unlikely to trigger a failure when their outputs can clarify environment behavior. This is important for both the solo-memory baseline and RedExplorer because the dependencies targeted by MAS-Comp-Bench are not stated in the role or tool descriptions.

D Plan Format, Validation, and Feedback

A submitted plan is an ordered sequence of actions. Each action names a tool and provides its typed arguments. An argument may be a literal permitted by the selected interface or a reference to a field returned by an earlier successful action. Before mutating environment state, the engine checks the plan structure, tool name, required fields, argument types, and reference direction. Forward or unresolved references are invalid.

After validation, the engine executes actions sequentially against a fresh environment instance for that attempt. For action a_t and state s_t, the transition function produces $S(a_t, s_t) \rightarrow (s_{t+1}, o_t, d_t)$, where o_t is the model-visible observation and d_t is structured evaluator data. The response returned to the red team records successful outputs or typed errors and, when applicable, the verified failure signature. Hidden state, provenance, and policy evidence are retained for replay and scoring but are not inserted into the next model prompt.

E Experimental Configurations

Table 9 collects the controls that differ across the primary comparison and ablations. Conditions within an experiment use the same environment instances and evaluated-plan budget. For RedExplorer, two execution agents produce two submitted attacks per orchestrator iteration; the number of orchestrator iterations is therefore halved relative to the solo baselines so that the total attack budget remains matched.

Table 9: Summary of the experimental configurations.

Experiment	Planner and models	Feedback / interface	Budget rule
Primary model comparison	Solo-memory; GPT-4o, Claude Opus 4.8, Qwen3.8-Max-0902, Kimi-K3, and Gemini 3.8 Flash	Interactive execution with the selected semantic-neutral interface	Matched evaluated-plan budget across models and environments
RedExplorer comparison	Memory, orchestration, and two execution agents; GPT-4o	Interactive execution with the same environment context and schemas as the solo baseline	Two attacks per orchestrator iteration; iterations halved to match total attacks
No-feedback control	Solo-cold; GPT-4o and Claude Opus 4.8	Environment responses suppressed; descriptions and schemas retained	Same proposal setting across the three environments
Interface ablation	Solo-cold GPT-4o	Opaque-literal, runtime-reference, and semantic-neutral interfaces	100 attempts per environment, five paired replicates per interface

F Metrics and Accounting

Table 10: Evaluation metrics used in the main experiments and ablations.

Metric	Definition
Valid plan rate	Fraction of submitted plans marked `valid_complete` or `valid_until_success` after schema and reference validation
Execution depth	Fraction of proposed actions successfully dispatched before termination, averaged over submitted plans
Strict mechanism recall	Fraction of reachable certified strict mechanisms discovered; macro-averaged across environments for aggregate reporting
Failure families	Number of distinct catalog families represented among the certified discoveries
Token usage	Sum of input and output tokens across all model calls attributable to the evaluated method
Cost	Token usage converted with the model- and provider-specific price schedule used for the run

An attack attempt is one submitted candidate plan, regardless of how many tool calls it contains. Invalid plans consume an attempt because they use the same model interaction budget as executable plans. If execution reaches a certified failure before the end of the proposed sequence, the plan may terminate with status `valid_until_success`; otherwise, successful execution of the full sequence receives `valid_complete`. Repeated trajectories that trigger the same catalog identity count once toward strict mechanism recall.

G Deterministic Replay and Reproducibility

Every evaluated plan is stored together with its interface condition, resolved arguments, ordered observations, state-transition log, oracle verdict, and failure identity when present. Replaying the same validated plan from the same initial environment state applies the same transition and policy-oracle code, which enables audit of both successful discoveries and plans scored as safe. Aggregate analysis should additionally retain the model identifier, provider, replicate identifier, attempt budget, token counts, and price schedule so that recall and cost-efficiency summaries can be regenerated from the saved traces.

H PROVIDER-NEUTRAL PROMPT SKELETONS

The implementation separates the semantic content of a prompt from the provider-specific message wrapper. The following skeleton captures the common instruction supplied to a solo planner; serialized tool schemas, observations, and memory are inserted into the marked fields.

> **Role.** Act as an automated red-team planner evaluating a multi-agent environment for failures that arise from interactions among locally valid actions.
> **Available information.** Use only the supplied environment description, agent roles, tool schemas, prior-attempt summary, and model-visible execution feedback. Do not assume access to hidden state or evaluator policies.
> **Task.** Propose one ordered tool-call plan that either tests an unknown dependency or exploits a dependency supported by earlier observations. Later actions may consume typed fields returned by earlier successful actions.
> **Output.** Return only the structured plan expected by the validator, including the tool name and typed arguments for every action.

RedExplorer retains this common environment contract while narrowing each agent's responsibility. The memory agent is instructed to compress raw history into observed dependencies, failed hypotheses, and promising follow-up probes, without producing an attack plan. The orchestration agent converts that summary into multiple strategically distinct composition paths and is asked to avoid duplicating the same tool family or failure hypothesis across paths when an alternative is available. Each execution agent receives exactly one path and is instructed to instantiate it as a concrete, schema-valid action sequence rather than revising the high-level objective. Full resolved prompts and raw model responses should be retained with the run artifacts so that formatting and parsing behavior can be audited alongside benchmark outcomes.