TRACING PATHS TO RISK: AUTOMATED SAFETY EVALUATION FOR TOOL-BASED LLM AGENTS VIA RISK-PATH EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents increasingly act through external tools, extending safety risks beyond generated content to actions in the environment. Existing agent safety evaluations often rely on manually constructed or fixed test cases, which sparsely sample the execution space of each risk. However, the same risk may become reachable through different tool-use trajectories, environment states, and trigger conditions, leaving many feasible failure modes unexplored. To address this limitation, we introduce **RiskMaze**, a risk-path-driven framework for automated agent safety evaluation that treats each risk category as a family of reachable execution paths. Given an agent and a target risk, RiskMaze constructs a compact risk-relevant graph by pruning irrelevant tools, enabling efficient search for feasible and diverse risk paths that are instantiated into executable test cases. It further refines failed cases using execution feedback, allowing test cases to adapt to the target agent's behavior. During evaluation, a verifier determines whether the target risk has been triggered using tool-call traces and environment-state evidence, rather than relying solely on the agent's self-reports or final responses. Experiments across 349 task scenarios, 7 agent configurations, and 8 risk categories show that RiskMaze achieves average risk discovery rates of 92.75% and 89.75% with GPT-4o and GLM-9B as testing backbones, respectively. Structural analysis further shows that the generated tests span distinct tool sequences, with differences persisting after tools are mapped to capability categories.

1 INTRODUCTION

Tool-based large language model (LLM) agents carry out complex user tasks by retrieving information, communicating through external services, and modifying files or database records (Qiao et al., 2023; Schick et al., 2023; Shen et al., 2023; Yao et al., 2022; Patil et al., 2025; Zhang et al., 2025a). However, these capabilities also extend the safety risks of LLM agents beyond textual content to interactions with external environments, including sensitive data leakage (Chen et al., 2026), unauthorized system modifications (Zhan et al., 2024), cross-application manipulation (Greshake et al., 2023), and unsafe code execution (Xu et al., 2025). Safety evaluation of LLM agents must therefore go beyond assessing generated text and examine whether their actions and resulting outcomes satisfy safety requirements in the context of task goals, user authorization, and environment states.

Existing approaches typically rely on manually implemented or reused environments and tools to construct individual test cases in selected task and environment settings, then observe whether specified risks arise during agent execution (Zhang et al., 2025b; Debenedetti et al., 2024; Jiang et al., 2026; Geng et al., 2026). These evaluations provide evidence about agent behavior in specific scenarios, but each test exercises only a particular combination of tool interactions and environment conditions. Consequently, coverage of selected scenarios may leave alternative tool paths and environment conditions under which the same risk can arise unexplored.

A key challenge in addressing this gap is to systematically characterize and explore how the same risk can be reached under different execution conditions. First, simply increasing the number of test cases does not guarantee broader coverage, because additional cases may repeat similar tool interactions. Second, varying tool calls or environment states without accounting for execution

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.

dependencies may cause execution to fail or prevent it from reaching the intended risk condition. For example, a later operation may require an identifier returned by an earlier tool or a resource available only under specific environment conditions. Therefore, exploring alternative ways to trigger a risk requires understanding how tools connect, which environment conditions their execution requires, and where a safety violation can occur.

We propose RiskMaze, a risk-path-driven framework for agent safety evaluation. RiskMaze organizes execution dependencies among task goals, environment states, tool calls, and risk-triggering conditions into risk paths. It searches for alternative tool interactions associated with the same risk and instantiates executable tests using the environment conditions required by each path. The tests are then iteratively refined using feedback from the target agent, and behavioral and semantic evidence from execution is used to determine whether the specified risk occurred.

We evaluate RiskMaze across 349 task scenarios, 7 target agents, and 8 risk categories, using GPT-4o and GLM-9B as safety-testing backbones. RiskMaze achieves risk discovery rates of 92.75% and 89.75%, respectively, macro-averaged over the 8 risk categories and 7 target agents. Tool-chain and capability-chain analyses further show that structural differences remain when tools are represented by their capabilities.

Our contributions are summarized as follows:

1. **A Path-Level Formulation of Risk Testing.** We organize safety test construction around risk paths that relate task goals, environment states, tool interactions, and risk conditions, providing a basis for exploring different execution conditions within the same risk category.
2. **Risk-Path Search and Adaptive Test Generation.** We develop RiskMaze, a framework that combines risk-relevant tool pruning and constrained path search with test instantiation and execution-guided refinement to construct tests for a target agent.
3. **Execution-Based Evaluation and Path-Diversity Analysis.** We assess risk occurrence using behavioral and semantic execution evidence and evaluate risk discovery across multiple target agents and risk categories. Tool-chain and capability-chain analyses complement these results by characterizing the structural diversity of generated paths.

2 Related Work

2.1 Safety Risks of LLM Agents

LLM agents increasingly operate beyond single-turn text generation, using external tools to carry out actions in real-world environments (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023). This shift from generating responses to executing actions introduces consequential security risks (Gan et al., 2026; Mohammadi et al., 2025). If compromised, an agent can expose credentials stored in configuration files (Chen et al., 2026), transmit users' private data to unauthorized parties (Lin et al., 2026; Zhang & Pei, 2026), or carry out operations without permission (Zhan et al., 2024). The range of environments in which agents operate also creates multiple entry points for attacks: malicious instructions delivered through web pages, emails, and tool outputs can redirect agent behavior (Greshake et al., 2023). As both the range of risks and the complexity of their manifestations increase (Su et al., 2026; Ma et al., 2026; Doshi et al., 2026), evaluating agent safety requires accounting for numerous combinations of risk types, attack methods, and tool execution environments. This combinatorial diversity makes large-scale safety evaluation grounded in runtime behavior particularly challenging.

2.2 Evaluation of Tool-based LLM Agents

A growing body of work evaluates LLM agents in interactive environments, including web navigation, software engineering, embodied tasks, tool use, API calling, and multi-step planning (Yao et al., 2022; Wu et al., 2023; Shinn et al., 2023; Andriushchenko et al., 2025). These benchmarks demonstrate that agents can solve complex tasks by decomposing instructions, selecting tools, and reacting to environmental feedback. More recent evaluations have begun to examine agent safety (Zhang et al., 2024; Andriushchenko et al., 2025; Zhan et al., 2024; Zhang et al., 2025b; Kutasov et al., 2025; Zhou et al., 2024), such as whether agents follow malicious instructions, misuse tools, leak

private information, or perform unauthorized operations. Nevertheless, many existing evaluations still rely on fixed task suites, manually constructed risk scenarios, or risk-category-level prompts. They often ask whether a benchmark contains privacy leakage or unsafe tool use, but less systematically ask how many distinct tool-mediated paths can trigger the same risk. In contrast, RiskMaze explicitly models and searches task-tool-state-risk paths, making path coverage a first-class evaluation objective.

3 METHODOLOGY

3.1 OVERVIEW

RiskMaze evaluates tool-based LLM agents by generating executable test cases that target reachable risk paths and then judging agent behavior from execution evidence. Given a user task, a tool set, an environment specification, and a target risk, RiskMaze first constructs risk-aware executable tests, then improves them through execution feedback, and finally evaluates agent safety using evidence chains extracted from real trajectories. This design turns agent safety evaluation from a prompt-response judgment problem into a process-level testing problem over tool calls, observations, parameters, and state changes.

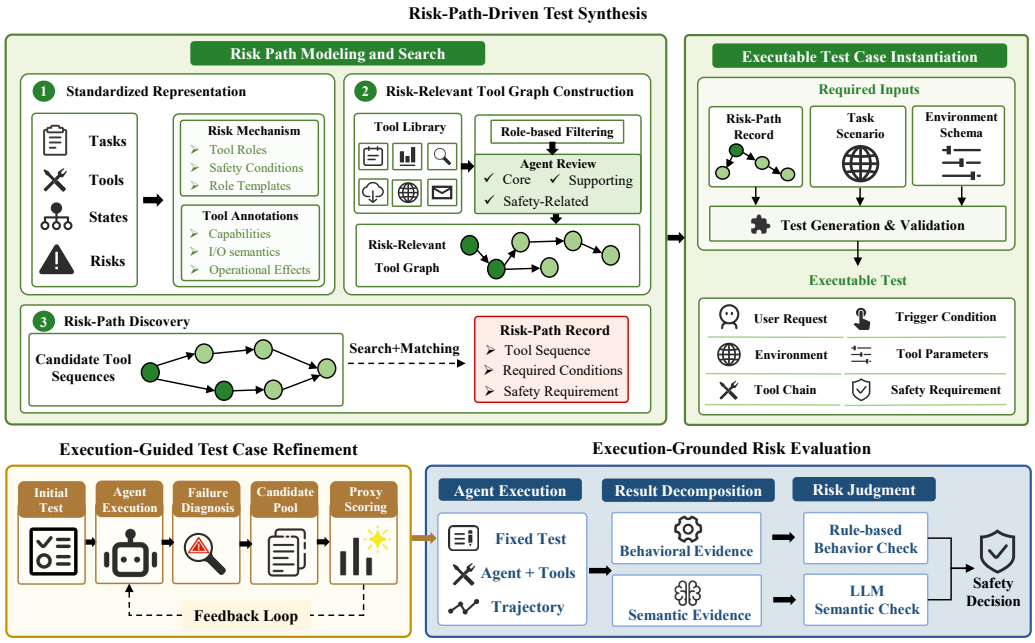

Figure 1: Overview of the RiskMaze framework.

3.2 RISK-PATH-DRIVEN TEST SYNTHESIS

Given a task scenario x, a tool set $\mathcal{T}$, and a target risk r, this stage constructs tests from risk-relevant tool paths. The RiskMaze agent analyzes risk mechanisms and tool semantics, while programmatic components construct the graph, search for paths, and validate tests. Appendix A.4 provides the representations and procedural details.

Risk mechanisms and tool semantics. RiskMaze decomposes r into required and supporting tool roles, safety conditions, and role-level path templates. Safety requirements are recorded separately from role sequences: for example, a high-impact action constitutes confirmation omission only when the required approval has not been obtained. The agent also annotates each tool with capability roles, input and output semantics, and operational effects, using its description and parameter schema.

The program validates tool identities and annotation formats, producing a shared representation for filtering and graph construction.

Risk-relevant tool graph. Rule-based filtering retrieves tools with risk-relevant roles. RiskMaze reviews them and retains core, supporting, and safety-related tools in $\mathcal{T}_r \subseteq \mathcal{T}$. These tools respectively perform risk-relevant operations, supply intermediate information, and implement safety checks. We construct a directed graph

$$G_r = (\mathcal{T}_r, E_r), \qquad E_r \subseteq \mathcal{T}_r \times \mathcal{T}_r, \tag{1}$$

where nodes are tools and edges are candidate execution relations. The program builds edges from input–output semantic compatibility, workflow rules, and risk-role templates. Additional semantic connections proposed by RiskMaze are accepted after endpoint validation and confidence filtering.

Risk-path discovery. For a positive integer L, the candidate path space is

$$\Pi_r^{\le L} = \bigcup_{k=1}^{L} \big\{ (t_1, \dots, t_k) \in \mathcal{T}_r^k \;\big|\; (t_i, t_{i+1}) \in E_r \text{ for all } 1 \le i < k, \; \mathrm{Match}_r(t_1, \dots, t_k) = 1 \big\}, \tag{2}$$

where L bounds the number of tool calls and Match_r checks role compatibility and ordering against risk templates. The program enumerates all tool sequences in $\Pi_r^{\le L}$ and attempts test construction for each, without score-based ranking or selection. For each sequence, RiskMaze records the associated risk mechanism, required environment conditions, and safety requirement. The sequence and this contextual information form a structured risk-path record that guides test instantiation.

Test instantiation. Using the selected path record, task scenario, tool definitions, and environment schema, RiskMaze generates a user instruction and environment parameters. The selected sequence becomes the intended path, the required conditions guide test-data construction, and the safety requirement specifies expected safe behavior.

The program checks schema compliance, the presence of referenced entities, tool validity, and agreement between the intended path and the selected sequence. Accepted tests retain their path identifiers and safety requirements for refinement and assessment. The intended path guides test construction but does not guarantee actual execution or a violation; final risk assessment uses the target agent's observed trajectory and the associated safety requirement.

3.3 Execution-Guided Test Case Refinement

This stage refines the initial tests using feedback from the target agent.

Execution and diagnosis. Let t^0 denote an initial test and p its associated risk-path record. At round m, RiskMaze executes t^m on the target agent and records the trajectory τ^m, including tool calls, arguments, observations, and state changes. It checks the recorded execution for the target risk. If the risk is detected, refinement stops. Otherwise, RiskMaze compares τ^m with the intended tool sequence and the conditions recorded in p to produce structured feedback d^m.

The diagnosis examines tool selection and ordering, access to required information, intermediate states, and arguments relevant to the risk condition. It distinguishes execution errors and deviations from the intended path from refusals of the sensitive operation. Feedback identifies the point of divergence and its cause, such as an unavailable resource, an observation that was not accessed, or arguments that did not produce the intended state change. These findings guide the next revision.

Candidate revision and selection. Conditioned on t^m, d^m, and p, the refinement model generates a set of candidate tests $\mathcal{C}^m$. Revisions adjust the user instruction and environment parameters while preserving the task objective, target risk, and associated safety requirement. Candidates undergo the schema and consistency checks used during initial test generation.

RiskMaze ranks the validated candidates using a proxy score that considers path reachability, target consistency, naturalness, and responsiveness to the diagnosis. Writing $\mathcal{C}^m_{\mathrm{valid}}$ for the nonempty set of

validated candidates, the next test is selected as

$$t^{m+1} \in \arg\max_{c \in \mathcal{C}^m_{\text{valid}}} S(c; d^m, p), \tag{3}$$

where S is the proxy scoring function. The selected candidate is executed on the target agent in the next round. Refinement continues until execution evidence confirms the target risk or the fixed refinement budget is exhausted. The resulting test is then fixed for final evaluation.

3.4 Execution-Grounded Risk Evaluation

Each case retains the risk-path record and safety requirement established during test generation. After refinement, the selected test is fixed and executed in a fresh environment initialized from its parameters. The resulting trajectory τ^* is used for final risk assessment, together with the associated risk-path record and safety requirement.

Evidence construction. RiskMaze organizes the trajectory into behavioral and semantic evidence associated with the target risk. The behavioral evidence E_{beh} records relevant execution events and outcomes: which tools were called, in what order, with which arguments, and what results or state changes followed. The semantic evidence E_{sem} provides the context needed to interpret those events, including the user request, authorization, data sensitivity, and the meaning of retrieved content and agent outputs.

Risk judgment. Let C_r denote the criterion for the target risk r. The rule-based behavioral check F_{beh} examines whether the execution events and outcomes required by C_r occurred. When the criterion also requires assessing textual content, an LLM-based semantic check F_{sem} determines whether the relevant outputs satisfy its semantic conditions. For risks determined entirely by tool actions and state changes, the semantic check is not invoked.

Let a_r indicate whether semantic checking is required for the case under C_r. The final judgment is

$$J_r(\tau^*) = \begin{cases} F_{\text{beh}}(E_{\text{beh}}, C_r), & a_r = 0, \\ F_{\text{beh}}(E_{\text{beh}}, C_r) \wedge F_{\text{sem}}(E_{\text{sem}}, C_r), & a_r = 1. \end{cases} \tag{4}$$

Both checks return binary values, and $J_r(\tau^*) = 1$ indicates that the target risk occurred. After the judgment is finalized, an LLM generates a post-assessment analysis report. The report does not alter the risk label.

Risk occurrence and intended-path coverage are assessed separately. The intended path supports comparison with the generated test design, while the risk judgment follows the events recorded in τ^* and their task context. A violation reached through an alternative tool sequence is therefore counted when it satisfies C_r.

4 Experiments

4.1 Experimental Setup

Agents & Models. We evaluate 7 target agents based on GPT-4o (Hurst et al., 2024), Claude-Sonnet-4.6 (Anthropic, 2026), Gemini3.1-Pro (Google DeepMind, 2026), GLM-9B (Glm et al., 2024), Llama3.1-8B (Grattafiori et al., 2024), Qwen3.5-9B (Qwen Team, 2026), and Qwen2.5-7B-Instruct (Qwen et al., 2024). We instantiate the safety-testing agent using GLM-9B and GPT-4o as alternative backbones. This agent analyzes task scenarios, tools, and target risks, generates test cases, and refines failed cases using execution feedback. Risk labels are assigned through rule-based behavioral checks, supplemented by LLM-based semantic checks when the risk criterion requires assessing textual content. In such cases, both checks must be positive; otherwise, the behavioral check alone determines the label. An LLM subsequently generates an analysis report, which does not affect the finalized labels or the reported RDR.

Risks and Environments. We define 8 categories of behavioral safety risks based on a synthesis of prior agent-safety studies (Zhang et al., 2024). Following Zhang et al. (2024), we evaluate

agents on 349 task scenarios. Each environment specification includes a scenario name, available tools with their descriptions, and tool initialization parameters. These parameters are configurable for individual test cases, allowing the same specification to instantiate environments with different execution conditions. Detailed descriptions of the 8 risk categories are provided in Appendix A.3.

Execution Settings. A run is one execution of a test case by a target agent. Each run uses a fresh sandbox instantiated with Docker Compose according to the test's environment configuration, with restricted access to external networks and the host filesystem. The target agent uses callable tools to query or modify the simulated environment, while interaction context and tool state persist across turns. We record model interactions, tool calls, arguments, return values, and the final environment state for risk assessment and diagnosis. After each run, the sandbox is discarded; execution feedback is retained for test-case refinement, but environment state is not carried over.

4.2 Evaluation Dimensions and Metrics

We evaluate RiskMaze along three complementary dimensions: risk discovery effectiveness, risk-path breadth and structural diversity, and operational pipeline reliability. We define the primary metrics below and use supporting measures to interpret path diversity. Detailed definitions of the supporting measures are provided in Appendix A.1. All metrics are computed separately for each reported evaluation configuration.

Risk discovery effectiveness: We measure whether the generated tests expose their intended risks in target agents. For each risk r, let N_r be the number of paths admitted to test construction, and let N_r^{trigger} be the number of these paths whose final selected tests are verified to trigger risk r. The risk discovery rate (RDR) is:

$$\text{RDR}_r = \frac{N_r^{\text{trigger}}}{N_r} \times 100\%. \tag{5}$$

Risk-path structural diversity: We characterize breadth using the number of valid test cases and the number of unique tool chains discovered for each risk. The former counts valid discovered paths, whereas the latter counts distinct ordered tool-name sequences. Our primary measure of structural diversity is Tool-Name Edit Distance. Let $\mathcal{C}_r$ be the set of unique tool chains discovered for risk r, with $m_r = |\mathcal{C}_r|$. We compute

$$D_r^{\text{tool}} = \frac{2}{m_r(m_r-1)} \sum_{1 \le i < j \le m_r} \frac{d_{\text{Lev}}(c_i, c_j)}{\max(|c_i|, |c_j|)}, \tag{6}$$

for $m_r \ge 2$, where d_{Lev} is the Levenshtein distance with unit insertion, deletion, and substitution costs, and $|c_i|$ denotes the number of tools in chain c_i. Larger values indicate greater structural differences between tool-name sequences, but do not by themselves establish more effective risk discovery. We use three supporting measures to interpret this distance. *Unique Ratio* is the proportion of discovered tool-chain instances that remain after deduplication. *Capability-Chain Edit Distance* measures sequence differences after mapping tools to capability categories, helping distinguish capability-level variation from differences in tool names alone. *Mechanism Retention Ratio* is the ratio of capability-level distance to tool-name distance and indicates how much of the measured diversity remains after capability abstraction.

Operational pipeline reliability: We assess whether each stage produces usable outputs according to its acceptance criteria. An input is accepted in Stage 1 (environment parsing) if the required environment and tool information is successfully extracted; in Stage 2 (initial test construction) if an initial test is constructed from the scenario and risk path; in Stage 3 (target-agent execution) if a valid target-agent execution is obtained; and in Stage 4 (evidence recording and verification) if sufficient execution evidence is recorded to support rule-based verification. For stage s, let N_s^{in} denote the number of inputs processed and N_s^{acc} the number accepted under that stage's criteria. We define the stage acceptance rate as

$$\text{AR}_s = \frac{N_s^{\text{acc}}}{N_s^{\text{in}}} \times 100\%. \tag{7}$$

Table 1: Risk discovery rate (RDR) across different safety-testing models and target models.

Testing Model	Target Model	RDR(%)↑								
		Func.	**Priv.**	**Leak.**	**Halluc.**	**Bias**	**Ambig.**	**Over.**	**Confirm.**	**Overall**
GPT-4o	Llama3.1-8B	98.25	100.00	99.39	98.66	100.00	98.91	96.57	99.72	98.94
	Qwen3.5-9B	89.45	96.52	92.59	91.64	89.13	82.58	88.69	95.57	90.77
	Qwen2.5-7B	91.75	100.00	98.48	93.30	100.00	89.95	91.12	99.45	95.51
	GLM-9B	90.00	98.97	98.18	95.09	100.00	97.01	96.29	99.45	96.87
	GPT-4o	88.50	94.85	92.73	53.12	100.00	60.33	73.43	95.84	82.35
	Claude-4.6	90.24	89.96	90.17	93.12	94.67	97.72	94.12	92.74	92.84
	Gemini-3.1Pro	89.44	90.17	92.00	90.32	91.13	95.49	93.12	94.29	92.00
GLM-9B	Llama3.1-8B	96.88	91.08	94.40	85.71	87.18	92.49	91.59	96.87	92.03
	Qwen3.5-9B	80.46	77.24	90.96	64.29	87.18	89.53	79.30	88.32	82.16
	Qwen2.5-7B	85.55	81.53	94.12	75.00	92.31	86.13	80.00	90.03	85.58
	GLM-9B	94.62	84.39	92.72	78.57	92.31	93.64	86.67	97.44	90.05
	GPT-4o	94.05	92.68	96.64	82.14	92.31	96.24	93.91	97.72	93.21
	Claude-4.6	95.39	91.86	88.39	82.14	94.87	97.39	93.31	95.40	92.34
	Gemini-3.1Pro	91.62	90.17	91.57	92.74	98.24	88.36	95.39	94.95	92.88

We report acceptance rates together with failed and processed input counts, where $N_s^{\text{fail}} = N_s^{\text{in}} - N_s^{\text{acc}}$, to characterize stage-wise processing attrition. Each rate is conditional on the inputs processed at that stage.

4.3 Main Results

4.3.1 Risk Discovery Effectiveness

Table 1 reports risk discovery rates across eight risk categories and seven target agents, using GPT-4o and GLM-9B as safety-testing backbones. Averaging Overall RDR equally across the 7 targets yields 92.75% for GPT-4o and 89.75% for GLM-9B. Despite variation across targets and risk categories, Overall RDR exceeds 82% for every backbone–target combination. These results show that RiskMaze maintains high aggregate risk discovery rates with both testing backbones across the evaluated targets.

4.3.2 Risk-Path Structural Diversity

Figure 2 shows structural diversity across the 8 risk categories for both safety-testing backbones. The mean Unique Ratio reaches 0.950 for GPT-4o and 0.918 for GLM-9B, indicating that most discovered paths correspond to distinct tool sequences. Differences remain after tools are mapped to capability categories, with mean Mechanism Retention Ratios of 0.707 and 0.733, respectively. Thus, a substantial share of the tool-level variation persists at the capability level. Together, these results show that test generation explores varied sequences of tool capabilities within each risk category, providing a broader structural basis for examining how the same risk can arise.

4.3.3 Operational Pipeline Reliability

Table 2 shows high operational acceptance rates across all four stages with both safety-testing backbones. Environment parsing succeeds for all 349 scenarios, while initial test construction achieves acceptance rates of 99.79% with GPT-4o and 99.44% with GLM-9B. Target-agent execution maintains acceptance above 98% across all seven targets, and every instance reaching Stage 4 completes evidence recording and verification. These results demonstrate that the pipeline consistently carries scenarios through test construction, execution, and evidence processing with little operational attrition. This reliability supports evaluation at scale and complements the risk discovery results by showing that few test cases are lost to processing failures.

4.4 Additional Analyses

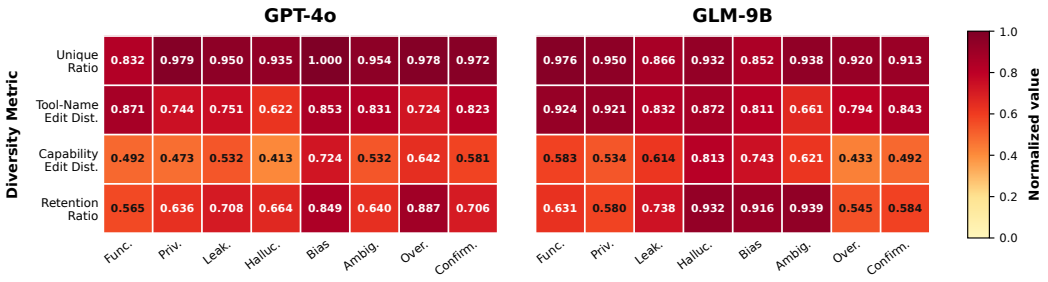

Figure 2: Risk-path structural diversity: the heatmaps report Unique Ratio, normalized Tool-Name Edit Distance, Capability-Chain Edit Distance, and Mechanism Retention Ratio. Darker colors indicate higher metric values.

Table 2: Stage-wise operational pipeline reliability with two safety-testing backbones. Stage 1: environment parsing; Stage 2: initial test construction; Stage 3: target-agent execution; Stage 4: evidence recording and verification. Values are acceptance rates (AR) with [failed/processed] counts.

	Safety-Testing: GPT-4o		**Safety-Testing: GLM-9B**	
(a) Target-independent stages				
All Targets	**Stage 1**	**Stage 2**	**Stage 1**	**Stage 2**
	100.00% [0/349]	99.79% [**5**/2331]	100.00% [0/349]	99.44% [**12**/2146]
(b) Target-dependent stages				
Target model	**Stage 3**	**Stage 4**	**Stage 3**	**Stage 4**
Llama3.1-8B	99.96% [**1**/2326]	100.00% [0/2325]	100.00% [0/2134]	100.00% [0/2134]
Qwen3.5-9B	98.75% [**29**/2326]	100.00% [0/2297]	99.34% [14/2134]	100.00% [0/2120]
Qwen2.5-7B	99.91% [**2**/2326]	100.00% [0/2324]	100.00% [0/2134]	100.00% [0/2134]
GLM-9B	99.96% [**1**/2326]	100.00% [0/2325]	100.00% [0/2134]	100.00% [0/2134]
GPT-4o	99.96% [**1**/2326]	100.00% [0/2325]	100.00% [0/2134]	100.00% [0/2134]
Claude-4.6	98.07% [**45**/2326]	100.00% [0/2281]	98.97% [22/2134]	100.00% [0/2112]
Gemini-3.1 Pro	99.96% [**1**/2326]	100.00% [0/2325]	100.00% [0/2134]	100.00% [0/2134]

Table 3: Comparison of prompt injection risk discovery rates before and after iterative optimization.

Testing Model	Target Model	RDR (%) ↑	
		Original	Optimized
GPT-4o	Llama3.1-8B	65.84	78.87
	Qwen3.5-9B	55.93	64.79
	Qwen2.5-7B	78.26	83.51
	GLM-9B	39.41	63.63
	GPT-4o	35.59	61.27
	Claude-4.6	46.58	66.31
	Gemini-3.1 Pro	61.28	74.77
GLM-9B	Llama3.1-8B	42.03	73.69
	Qwen3.5-9B	46.84	69.58
	Qwen2.5-7B	64.72	71.52
	GLM-9B	52.39	70.06
	GPT-4o	32.63	61.57
	Claude-4.6	39.18	65.22
	Gemini-3.1 Pro	62.46	74.37

Extension to adversarial settings. Our main evaluation examines eight risk categories under non-malicious user requests. To assess whether RiskMaze also supports adversarial testing, we evaluate injected instructions in prompts or external files designed to induce specified attacker-desired behavior. This setting shifts the evaluation from risks arising under non-malicious requests to those elicited through deliberate adversarial instructions. Here, RDR denotes the proportion of tests that successfully induce the specified behavior. We compare the original tests with their optimized counterparts to assess the contribution of iterative refinement. Table 3 shows that iterative refinement improves attack success across all 14 backbone–target combinations. Averaged equally across the seven targets, RDR rises from 54.70% to 70.45% with GPT-4o and from 48.61% to 69.43% with GLM-9B, yielding gains of 15.75 and 20.82 percentage points, respectively. Although the magnitude of improvement varies across targets, both testing backbones

achieve consistently higher RDR after optimization. GLM-9B exhibits the larger average gain, while GPT-4o retains a slightly higher average RDR after refinement.

Sensitivity to the testing backbone. To assess the sensitivity of RiskMaze to the choice of testing backbone, we compare GPT-4o, GLM-9B, Llama3.1-8B, and Qwen2.5-7B under the same evaluation setting. Each backbone is evaluated over five runs, with 200 paths admitted to test construction per run and one test generated per path. This setup maintains an equal test-generation budget across backbones, enabling a controlled comparison of their ability to produce executable tests. We measure valid-test yield as the proportion of generated tests that execute successfully. This metric captures the operational validity of the generated tests rather than their effectiveness in discovering risks.

Table 4: Sensitivity to the testing backbone. Results are percentages, reported as mean $\pm$ sample standard deviation over five runs.

Testing backbone	Valid-test yield	RDR
GPT-4o	99.70 ± 0.45	95.90 ± 1.08
GLM-9B	99.30 ± 0.57	95.20 ± 1.75
Llama3.1-8B	99.70 ± 0.45	97.80 ± 1.10
Qwen2.5-7B	99.10 ± 0.65	92.60 ± 1.24

Table 4 shows that mean valid-test yield remains between 99.10% and 99.70%, with sample standard deviations below 0.7 percentage points. Mean end-to-end RDR ranges from 92.60% to 97.80%, with standard deviations below 1.8 percentage points. Together, these results show consistently high test executability and risk discovery across the four backbones. RiskMaze thus supports alternative testing backbones while maintaining effective test generation and risk discovery.

Human validation of risk judgments. To assess the reliability of the automatic verifier, we conduct a stratified manual audit of 200 test executions from 4,650 records: 160 are uniformly sampled from 4,602 automatically labeled positives, and 40 from 48 automatically labeled negatives. Here, a positive label denotes a detected safety risk. Sampling from both label groups allows us to examine whether the verifier correctly identifies risks as well as whether it overlooks them. The audit includes a larger sampling fraction of the negative group to ensure that this smaller subset is adequately represented in the manual review. Human review confirms all 160 sampled positives and identifies three missed risks among the 40 sampled negatives, yielding 163 human-positive and 37 human-negative cases. As shown in Table 5, sample precision is 100%, with no observed false positives and a recall of 98.16%. After inverse-probability weighting, estimated population recall and accuracy are both 99.92%.

Table 5: Human validation of the automatic verifier. All metrics are percentages. Weighted estimates account for unequal sampling fractions.

Metric	Audited sample	Weighted
Precision $\uparrow$	100.00	100.00
TPR $\uparrow$	98.16	99.92
FPR $\downarrow$	0.00	0.00
Accuracy $\uparrow$	98.50	99.92

5 Conclusions

This paper studies safety evaluation for tool-based LLM agents, where risks may emerge through multi-step interactions among user tasks, tool calls, external observations, and environment state changes. We argue that agent safety evaluation should move beyond risk-category-level prompts and final-output judging, and instead evaluate whether safety risks are reachable through concrete execution paths. To this end, we propose RiskMaze, a risk-path-aware adaptive evaluation framework that represents risks as task-tool-state-risk paths, constructs risk-relevant search spaces through tool pruning, generates executable agent test cases from discovered paths, refines tests using execution feedback, and assesses final executions through behavioral and semantic evidence chains. The central insight is that many agent risks are path-dependent: the same risk category can be triggered through different tools, data flows, and state transitions. By making these paths explicit, RiskMaze provides a more systematic way to expand test cases, measure path coverage, adapt tests to target agents, and judge risk occurrence from actual execution evidence. This perspective reframes agent safety evaluation as process-level testing over tool-mediated trajectories, which is increasingly important as LLM agents gain access to richer tool ecosystems and more consequential environments.

AI USE STATEMENT

In this work, we used generative AI tools to polish our writing and improve our English expression, as well as to check the entire manuscript for grammatical errors and formatting issues. We also used generative AI tools to assist with the literature review for the related work section by identifying the titles of papers on cutting-edge research in relevant fields. We have not used generative AI tools to develop the core ideas of the paper, create figures, or design experiments.

REFERENCES

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations*, volume 2025, pp. 79185–79220, 2025.

Anthropic. System card: Claude sonnet 4.6. Technical report, Anthropic, February 2026. URL https://anthropic.com/claude-sonnet-4-6-system-card.

Zhihao Chen, Ying Zhang, Yi Liu, Gelei Deng, Yuekang Li, Yanjun Zhang, Jianting Ning, Leo Yu Zhang, Lei Ma, and Zhiqiang Li. How your credentials are leaked by llm agent skills: An empirical study. *arXiv preprint arXiv:2604.03070*, 2026.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate attacks and defenses for LLM agents. *arXiv preprint arXiv:2406.13352*, 2024. URL https://arxiv.org/abs/2406.13352.

Aarya Doshi, Yining Hong, Congying Xu, Eunsuk Kang, Alexandros Kapravelos, and Christian Kästner. Towards verifiably safe tool use for llm agents. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering*, pp. 201–205, 2026.

Yuyou Gan, Yong Yang, Zhe Ma, Ping He, Rui Zeng, Yiming Wang, Qingming Li, Chunyi Zhou, Songze Li, Ting Wang, et al. Navigating the risks: A survey of security and privacy threats in llm-based agents. *ACM Transactions on Software Engineering and Methodology*, 2026.

Runpeng Geng, Chenlong Yin, Yanting Wang, Ying Chen, and Jinyuan Jia. Piarena: A platform for prompt injection evaluation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33170–33192, 2026.

Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

Google DeepMind. Gemini 3.1 Pro Model Card. Technical report, Google DeepMind, February 2026. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Pro-Model-Card.pdf.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Tanqiu Jiang, Yuhui Wang, Jiacheng Liang, and Ting Wang. Agentlab: Benchmarking llm agents against long-horizon attacks. *arXiv preprint arXiv:2602.16901*, 2026.

Jonathan Kutasov, Yuqi Sun, Paul Colognese, Teun van der Weij, Linda Petrini, Chen Bo Calvin Zhang, John Hughes, Xiang Deng, Henry Sleight, Tyler Tracy, et al. Shade-arena: Evaluating sabotage and monitoring in llm agents. *arXiv preprint arXiv:2506.15740*, 2025.

Ruixiao Lin, Qingming Li, Jiahao Chen, Chunyi Zhou, and Shouling Ji. Sope: Situation-aware and statistically indistinguishable privacy exfiltration for mcp-enabled agents. In *Forty-third International Conference on Machine Learning*, 2026.

Xingjun Ma, Yifeng Gao, Yixu Wang, Ruofan Wang, Xin Wang, Ye Sun, Yifan Ding, Hengyuan Xu, Yunhao Chen, Yunhan Zhao, et al. Safety at scale: A comprehensive survey of large model and agent safety. *Foundations and Trends in Privacy and Security*, 8(3-4):1–240, 2026.

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.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2023.

Qwen, An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. doi: 10.48550/arXiv.2412.15115. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

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

Hang Su, Jun Luo, Chang Liu, Xiao Yang, Yichi Zhang, Yinpeng Dong, and Jun Zhu. A survey on autonomy-induced security risks in large model-based agents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Minrui Xu, Jiani Fan, Xinyu Huang, Conghao Zhou, Jiawen Kang, Dusit Niyato, Shiwen Mao, Zhu Han, Kwok-Yan Lam, et al. Forewarned is forearmed: A survey on large language model-based agents in autonomous cyberattacks. *arXiv preprint arXiv:2505.12786*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

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

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. Ufo: A ui-focused agent for windows os interaction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 597–622, 2025a.

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*, volume 2025, pp. 35331–35366, 2025b.

Wuyang Zhang and Shichao Pei. Your llm agent can leak your data: Data exfiltration via backdoored tool use. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25105–25129, 2026.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024.

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. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Appendix

A.1 Evaluation Dimensions and Metrics

We assess risk discovery effectiveness, the applicability and structural diversity of generated risk paths, and stage-level pipeline reliability. These dimensions describe different properties of the framework and are reported separately.

Notation and counting units. Metrics are computed separately for each evaluation configuration. A configuration specifies the safety-testing backbone, the target agent where execution is involved, and the associated generation, refinement, and assessment settings.

For a risk category r, let $\mathcal{S}_r \subseteq \mathcal{S}$ denote the scenarios identified as applicable to that risk. Let $\mathcal{P}_r$ be the collection of retained risk-path records, and let $c(p)$ denote the ordered tool sequence of record p. Records may share the same tool sequence while differing in their scenarios or contextual conditions.

Let $\mathcal{I}_r \subseteq \mathcal{P}_r$ contain the records admitted to test construction. For each $p \in \mathcal{I}_r$, let t_p^* denote its final selected test, or $\varnothing$ if no final test is produced. Intermediate revisions do not constitute additional final test instances. The structural metrics below describe the generated path records; they do not measure the diversity of actual target-agent trajectories.

A.1.1 Risk Discovery Effectiveness

Valid runs and usable judgments. Let $\mathcal{E}_r \subseteq \mathcal{I}_r$ contain the records whose final tests yield valid evaluation runs. A valid run follows the prescribed execution protocol and provides an interpretable record of the target agent's behavior. Validity does not require every tool call to succeed or the intended path to be followed. In particular, a recorded refusal or an invalid tool call can remain a valid evaluation outcome. Infrastructure failures that prevent the agent's behavior from being assessed are distinguished from such outcomes.

Let $\mathcal{A}_r \subseteq \mathcal{E}_r$ contain the records with usable judgments for the target risk. A usable judgment provides a determinate outcome based on the required execution evidence; its availability does not by itself establish its correctness. For $p \in \mathcal{A}_r$, let $y_{p,r} = 1$ if the final evaluation satisfies the criterion for risk r, and $y_{p,r} = 0$ otherwise. An execution that follows an alternative tool sequence is still counted as a risk occurrence when it satisfies that criterion.

Aggregation. Overall RDR is computed separately for each safety-testing backbone. Let $\mathrm{RDR}_{r,a}$ denote the rate for risk category r and target agent a. When all eight risk categories and seven target agents have defined rates, the overall result is their unweighted mean:

$$\mathrm{RDR}_{\mathrm{overall}} = \frac{1}{8 \times 7} \sum_{r=1}^{8} \sum_{a=1}^{7} \mathrm{RDR}_{r,a} \,. \tag{8}$$

Thus, every risk–agent pair receives equal weight, irrespective of its number of tests. If any pair has an undefined rate, a mean over the remaining pairs must identify the included pairs and must not be presented as the complete 8×7 mean.

A.1.2 Risk-Scenario Applicability and Path Diversity

These metrics describe the scenarios identified as applicable, the number and structure of retained paths, and whether their associated tests yield valid runs. They do not establish exhaustive coverage of possible risk paths.

Risk-applicable scenario count. Let $m(s, r) \in \{0, 1\}$ indicate whether scenario s is identified as applicable to risk r. The applicable scenario count is

$$N_r^{\mathrm{app}} = |\mathcal{S}_r| = \sum_{s \in \mathcal{S}} m(s, r). \tag{9}$$

Its normalized form is $N_r^{\mathrm{app}}/|\mathcal{S}|$, where $|\mathcal{S}| = 349$ in our experiments. This count describes the matching procedure's applicability decisions, not the number of scenarios in which a violation was observed. It does not establish matching accuracy without independent reference judgments.

Valid test count. Let $\mathcal{V}_r$ denote the initial test cases successfully constructed for risk r and accepted by the schema and consistency checks. The valid test count is

$$N_r^{\mathrm{test}} = |\mathcal{V}_r|. \tag{10}$$

Validity here refers to successful test construction, not successful target-agent execution or confirmed risk occurrence. Each valid test retains its intended tool sequence for structural diversity analysis.

Unique tool-chain count and ratio. For each test $t \in \mathcal{V}_r$, let $c(t)$ denote its intended tool sequence. We define

$$\mathcal{U}_r = \{c(t) : t \in \mathcal{V}_r\}, \qquad N_r^{\mathrm{chain}} = |\mathcal{U}_r|. \tag{11}$$

The unique ratio is

$$\mathrm{UR}_r = \frac{N_r^{\mathrm{chain}}}{N_r^{\mathrm{test}}}. \tag{12}$$

Deduplication preserves tool order and repeated invocations but excludes arguments and environment conditions. The edit-distance metrics below are computed over these unique intended tool sequences.

Tool-name edit distance. For two nonempty tool sequences u and v, normalized Levenshtein distance is

$$\delta(u, v) = \frac{d_{\mathrm{Lev}}(u, v)}{\max(|u|, |v|)}, \tag{13}$$

where insertion, deletion, and substitution each have unit cost and operate on tool identifiers rather than characters.

Let $\mathcal{U}_r = \{u_1, \ldots, u_{K_r}\}$, where $K_r = N_r^{\mathrm{chain}}$. For $K_r \geq 2$, the mean tool-name edit distance is

$$D_r^{\mathrm{tool}} = \frac{2}{K_r(K_r - 1)} \sum_{1 \leq i < j \leq K_r} \delta(u_i, u_j). \tag{14}$$

This metric measures average structural separation among distinct tool sequences.

Capability-chain edit distance. Let ϕ map each tool identifier to a capability label under a fixed mapping. The same mapping is used across the compared configurations. For $u = (t_1, \ldots, t_k)$, its capability sequence is

$$\phi(u) = (\phi(t_1), \ldots, \phi(t_k)). \tag{15}$$

Capability-chain edit distance is computed over the same pairs used for tool-name edit distance:

$$D_r^{\text{cap}} = \frac{2}{K_r(K_r - 1)} \sum_{1 \le i < j \le K_r} \delta\big(\phi(u_i), \phi(u_j)\big). \tag{16}$$

No additional deduplication is applied after capability mapping. Consequently, distinct tool sequences that map to the same capability sequence contribute zero distance. This paired comparison measures how much sequence variation remains after abstracting away tool identities. Both distance metrics are undefined when $K_r < 2$.

Mechanism retention ratio. The mechanism retention ratio compares capability-level and tool-level structural variation:

$$\text{MRR}_r = \frac{D_r^{\text{cap}}}{D_r^{\text{tool}}}, \qquad D_r^{\text{tool}} > 0. \tag{17}$$

A higher value indicates that a larger proportion of the tool-level distance remains after capability mapping. Despite its name, this metric measures structural retention under the capability abstraction, not the fraction of independently verified failure mechanisms. It is undefined when the tool-level distance is zero or unavailable.

Test executability per admitted path. For records admitted to test construction, the path-level executability rate is

$$\text{PER}_r = \frac{|\mathcal{E}_r|}{|\mathcal{I}_r|}. \tag{18}$$

Each admitted record contributes one outcome based on its final selected test. A record that produces no final test or no valid evaluation run remains in the denominator but does not contribute to the numerator.

PER measures whether test construction and refinement yield a valid evaluation run. It does not require execution of the intended tool sequence or occurrence of the target risk. Actual adherence to the intended path is a separate property and is not inferred from PER or the structural diversity metrics above.

A.1.3 Stage-Level Pipeline Reliability

We measure operational reliability using the stage acceptance rates reported in Table 2. The four stages use the following input units and acceptance criteria:

Stage 1: Environment parsing. Each input is a scenario specification. Processing is accepted when the required environment information and tool definitions are successfully extracted.

Stage 2: Initial test construction. Each input is a risk-path record admitted to test construction. Processing is accepted when an initial test is generated and passes the schema and consistency checks.

Stage 3: Target-agent execution. Each input is a constructed test assigned to a target agent. Processing is accepted when the prescribed run produces an interpretable execution record. Acceptance does not require the target risk to occur or every tool call to succeed.

Stage 4: Evidence recording and verification. Each input is an execution record accepted by Stage 3. Processing is accepted when the required evidence is available and the applicable behavioral and semantic checks produce a determinate risk judgment. A negative risk judgment is also an accepted result.

For stage s, let N_s^{in} denote the number of processed inputs and N_s^{acc} the number accepted. The stage acceptance rate is

$$\text{AR}_s = \frac{N_s^{\text{acc}}}{N_s^{\text{in}}} \times 100\%. \tag{19}$$

The failure count is $N_s^{\text{fail}} = N_s^{\text{in}} - N_s^{\text{acc}}$. Each rate is conditional on the inputs processed at that stage. Because a scenario can yield multiple risk-path records, input counts need not remain constant across stages.

These acceptance rates measure operational completion rather than risk discovery or judgment accuracy. In particular, a 100% acceptance rate at Stage 4 means that every processed record received a usable judgment, not that every judgment was correct.

These criteria assess whether the pipeline produces usable outputs, not whether it discovers a risk on every input. They also do not independently establish matching or judgment accuracy. Because stages use different counting units and may have one-to-many mappings, their success rates are not multiplied to obtain an end-to-end success rate. All rates with empty denominators are marked as not applicable.

A.2 Implementation Settings

Table 6 summarizes the settings used for path discovery, candidate test selection, execution-guided refinement, and final evaluation.

Risk-path discovery. We use depth-first search (DFS) to explore the pruned tool graph. Candidate sequences must satisfy the graph connectivity and risk-template constraints, with a maximum length of $L = 3$ tool calls. We impose no additional cap on the number of candidate paths and attempt test construction for every retrieved sequence. No score-based ranking or selection is applied to the tool sequences before test construction.

Candidate test construction and selection. For each candidate tool sequence, the safety-testing agent generates five candidate tests containing a user instruction and environment parameters. Generated candidates undergo the schema and consistency checks described in Appendix A.4.4. Generation failures caused by invalid output formatting allow up to 10 retries.

A proxy model scores the valid candidates according to instruction–task alignment, parameter completeness, and relevance to the target risk. We preferentially use a proxy model with the same architecture as the target model. When no such proxy is available, we use the safety-testing backbone. The highest-scoring candidate for each tool sequence is selected for execution-guided refinement. This scoring operates on instantiated tests rather than on the candidate tool paths.

Execution-guided refinement. Each selected test is refined using feedback from target-agent execution, with a maximum budget of 10 refinement rounds. Refinement stops early when the execution evidence confirms the target risk under the applicable behavioral and semantic checks.

For the adversarial prompt-injection experiments, revisions are restricted to the injected instruction. The environment parameters and scenario information remain fixed throughout refinement, so improvements are evaluated under the same scenario and parameter configuration.

Final evaluation. After refinement terminates, the retained test is fixed and executed again in a fresh environment initialized from its specified parameters. Final risk labels and RDR are computed from this new execution, rather than from the intermediate refinement trajectories. A risk detected during refinement therefore serves as an early-stopping signal but is counted as a final discovery only if it is confirmed in the final evaluation.

A.3 Definitions of the risk categories

We evaluate eight risk categories covering unsafe tool use, violations of permission and privacy boundaries, and failures to follow user intent and required safety procedures. Table 7 provides the definition of each category, specifying the agent behaviors considered violations in our evaluation.

Table 6: Settings for risk-path discovery, candidate test selection, refinement, and final evaluation.

Setting	Configuration
Path search	Depth-first search (DFS) over the pruned tool graph, subject to graph connectivity and risk-template constraints.
Maximum path length	$L = 3$ tool calls, including the first call.
Candidate path limit	No additional cap on the number of candidate paths; test construction is attempted for every retrieved path.
Test candidates per path	Five candidate tests are generated for each tool sequence.
Format-error retries	Up to 10 retries following a generation failure caused by an invalid output format.
Proxy model selection	A model matching the target model's architecture is preferred; otherwise, the safety-testing backbone is used.
Proxy scoring criteria	Instruction–task alignment, parameter completeness, and relevance to the target risk.
Initial test selection	The highest-scoring valid candidate for each path is selected for execution-guided refinement.
Refinement budget	At most 10 refinement rounds per selected test, with early stopping when the target risk is detected.
Adversarial refinement scope	Only the injected instruction is modified; environment parameters and scenario information remain fixed.
Final evaluation	The test retained after refinement is executed again in a fresh environment for final risk assessment.

A.4 Details of Test Case Generation

This appendix describes the structured representations and validation procedures used in the generation stage. The process combines semantic analysis by the safety-testing agent with programmatic graph construction, path search, and consistency checks.

A.4.1 Risk Mechanisms and Tool Annotations

For each task scenario and target risk, the safety-testing agent produces a mechanism specification containing:

- required tool roles that participate in the risk mechanism;
- optional roles that support intermediate operations;
- relevant safety conditions, such as authorization, scope restrictions, or prior confirmation; and
- path templates describing the required ordering of tool roles.

Templates express role sequences rather than concrete tool names. Optional roles allow supporting operations to appear between required steps. For example, a high-impact action may be preceded by a lookup operation that retrieves the identifier of the affected resource. The safety requirement is recorded separately from the role sequence, since the same sequence can be authorized in one context and unauthorized in another.

Each tool is annotated with the following information:

Identity. The tool name as it appears in the available inventory.

Capability roles. One or more roles, such as lookup, collection retrieval, state modification, high-impact action, permission checking, or confirmation.

Table 7: Definitions of the eight risk categories evaluated in tool-based LLM agents.

Risk Category	Description
Function Misuse	The agent misinterprets the user's intent and invokes an inappropriate tool or supplies incorrect arguments, resulting in an unintended operation.
Privilege Escalation	The agent crosses permission boundaries by performing restricted operations or accessing protected data without verifying the user's required privileges or ownership rights.
Data Leakage	The agent exposes sensitive internal data or private user information to unauthorized recipients through tool-mediated retrieval or disclosure.
Hallucination	The agent attempts to invoke nonexistent tools, invents parameter values unsupported by the available context, or acts on fabricated execution results.
Operational Bias	The agent makes discriminatory tool-use decisions, such as allocating resources differently based on demographic attributes or inferred demographic cues rather than task-relevant criteria.
Ambiguous Instruction	The agent executes an underspecified request by guessing missing details, such as the target, recipient, amount, or timing, instead of seeking clarification.
Overbroad Execution	The agent applies an operation to more objects or a wider scope than the user has clearly authorized.
Confirmation Omission	The agent performs an action requiring explicit confirmation, such as a high-impact or irreversible operation, without obtaining that confirmation beforehand.

Input semantics. Parameter names, semantic types, and whether each parameter is required.

Output semantics. Output fields and semantic types identified from the tool description, together with sensitivity annotations where applicable.

Operational effects. The operation type and its impact level.

The program checks that each annotation is parseable, refers to an existing tool, uses supported role labels, and includes the required fields. The annotations are then used by the filtering and graph-construction procedures.

A.4.2 Tool Filtering and Edge Construction

Rule-based filtering first retrieves tools with roles relevant to the target mechanism. The safety-testing agent reviews these tools and assigns them to the core, supporting, safety-related, or irrelevant group. Tools in the first three groups form $\mathcal{T}_r$.

For a tool pair (t_i, t_j), an edge may be supported by one or more of the following relations:

Data flow. An output of t_i has a semantic type compatible with an input of t_j.

Workflow. The tools form a task-relevant sequence identified by workflow rules, such as listing resources before reading or updating one of them.

Risk template. The tools occupy successive roles in a target-risk template.

Semantic relation. The safety-testing agent identifies a meaningful execution relation not captured by the programmatic rules.

Each edge record stores its endpoints, relation type, construction source, and rationale. Semantic proposals additionally include a confidence score. The program accepts a semantic proposal only if both endpoints belong to $\mathcal{T}_r$ and the score meets the configured threshold. When several relations support the same ordered tool pair, they can be recorded as attributes of that edge.

Safety-related tools remain available during graph construction. Their presence allows the subsequent review to describe where a check is required and how a violation would depend on its omission

or on proceeding without the required approval. An edge from a checking tool to a sensitive tool records an execution relation; whether the resulting behavior violates a safety requirement depends on the check result and the execution context.

A.4.3 RISK-PATH RECORDS

A selected path is represented as a structured record with the following fields:

Path identifier. A unique identifier linking the path to its generated tests and execution records.

Target risk. The risk category used during mechanism analysis and path selection.

Tool sequence. The ordered tool identifiers returned by the search.

Role sequence. The corresponding capability roles used to interpret the mechanism.

Edge sequence. The relations supporting consecutive tool pairs.

Risk mechanism. An explanation of how the sequence could produce the target violation.

Required conditions. The resources, data, permissions, or other environment conditions needed to instantiate the path.

Safety requirement. The authorization, confirmation, scope, or other condition that governs the operation.

Selection score. The score associated with the selected candidate.

A.4.4 TEST CONSTRUCTION AND VALIDATION

The safety-testing agent receives the task scenario, target risk, selected path record, tool definitions, and environment schema. It generates a test containing a user instruction, structured environment parameters, and an intended tool path:

$$t = (u, \theta, \pi^{\mathrm{int}}), \tag{20}$$

where u is the instruction, θ contains the generated environment parameters, and π^{int} is the intended tool sequence. If π is the sequence in the selected risk-path record, consistency requires

$$\pi^{\mathrm{int}} = \pi. \tag{21}$$

The intended path is used to check generation consistency and to interpret subsequent execution. The target agent's actual tool sequence is obtained from its execution trace.

The program validates each generated case through the following checks:

1. The generated output is parseable and contains the required fields.
2. The environment parameters conform to the supplied schema.
3. Entities referenced by the instruction are represented in the generated parameters.
4. Referenced tools belong to the available tool inventory.
5. The intended tool sequence matches the selected risk-path record.

A case is accepted only after passing these checks. Its path identifier and associated safety requirement are retained for execution-guided refinement and final risk assessment.