AUTOMATED DISCOVERY OF FAILURE MODES IN AGENTIC AI SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems composed of large language model-driven agents are increasingly used in *real-world* applications, yet evaluating their reliability remains challenging. Compared with single-model systems, their decentralized, multi-step, and stochastic behavior complicates failure analysis. We present a framework for automated identification of failure modes in agentic AI systems with limited or no historical interaction data. Our framework generates domain-specific evaluation scenarios, applies hierarchical failure abstraction, and uses statistical testing to separate systematic weaknesses from stochastic noise. We evaluate the framework on a real-world multi-agent customer-service system, demonstrating realistic failure-mode detection with minimal human effort. Results show that our framework identifies failure modes missed by capability benchmarks and LLM-judge-based evaluation approaches.

1 INTRODUCTION

Agentic artificial intelligence (AI) systems–composed of large language model (LLM)-based agents that interact with tools, environments, and users–are increasingly deployed in real-world applications such as customer service and enterprise automation (Park et al., 2025; Vishwakarma et al., 2025; Pan et al., 2025; Zhou et al., 2023; Wu et al., 2024). Compared to single-agent LLM systems, these systems involve long-horizon execution, multi-step orchestration, shared state, and dynamic interactions among multiple agents and tools, which substantially complicate failure analysis and maintenance.

Existing approaches to agent evaluation emphasize different aspects of system behavior (Yehudai et al., 2025). Benchmarks (Jimenez et al., 2023; Gupta et al., 2026; Li et al., 2026) focus on isolated agent capabilities under standardized tasks, adversarial testing and red-teaming probe safety and security vulnerabilities (Chen et al., 2024; Jiang et al., 2026; Zhang et al., 2024), and observability tools improve execution visibility through tracing and logging (Moshkovich et al., 2025; Deshpande et al., 2025). These directions have produced valuable insights but are not designed to systematically identify, prioritize, and mitigate recurring failure modes in complex multi-agent systems (MAS) as they are deployed in real-world settings. As a result, state-of-the-art evaluations often rely on task-level success metrics, capability-specific benchmarks, or single-agent proxies that treat failures as isolated outcomes rather than consequences of multi-agent interaction. This abstraction obscures recurring, coordination-driven, and stochastic failure patterns, motivating trace-level, failure-mode-oriented evaluation for effective diagnosis and iterative system improvement.

These challenges are especially pronounced in multi-agent and conversational settings (Barres et al., 2025; Levi & Kadar, 2025), where failures often arise from emergent agent interactions, tool dependencies, and long multi-turn dialogues under operational constraints. In practice, many such failures surface only through deployment or large-scale testing and must be analyzed retrospectively from execution traces, often with limited labels and high debugging cost. This leaves a gap between how agentic AI systems are evaluated in research settings and how failures are encountered, diagnosed, and mitigated in real-world deployments (Mohammadi et al., 2025; Mehta, 2025).

Agents in Production Pan et al. (2025) survey 306 practitioners across 26 domains and find that production agents are deliberately constrained: 68% execute at most 10 steps before human intervention, 70% use off-the-shelf models, 79% rely on manually constructed prompts, 74% depend

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.

primarily on human evaluation, and 75% forgo formal benchmarks in favour of A/B testing or expert feedback. These findings stand in sharp contrast to research directions emphasising highly autonomous agents in closed, benchmarked environments. While multi-agent systems are actively studied (Zhang et al., 2025a; Zhu et al., 2026; 2025), systematic failure-driven evaluation and improvement under realistic deployment constraints remain underexplored. MAP further identifies reliability–not maximal capability–as the primary production concern, motivating our evaluation and mitigation approach.

Benchmark Limitations for MAS Evaluation Agent benchmarks evaluate isolated single-agent capabilities (Liu et al., 2023; Mialon et al., 2023); extensions to multi-agent settings address only one property at a time–tool calling, planning, or collaboration (Guo et al., 2024; Geng & Chang, 2025; Yan et al., 2025; Sun et al., 2025; Wang et al., 2024)–and transfer poorly to multi-agentic system evaluation due to domain mismatch and interaction complexity (Chen et al., 2021; Merrill et al., 2026). For conversational MAS, behavior emerges from multi-turn bidirectional interactions that property-specific benchmarks cannot capture (Ma et al., 2026). More fundamentally, aggregate task-level success metrics misalign with deployment needs, where failures surface through execution traces and must be diagnosed retrospectively.

We address this gap with an end-to-end framework for detecting, validating, and mitigating failure modes in production-oriented MAS under limited historical data (Figure 1). Operating on execution traces, it produces statistically validated, hierarchical failure modes that drive targeted mitigation, improving task success from 74.9% to 85.3% through two rounds of automated mitigation with minimal human effort.

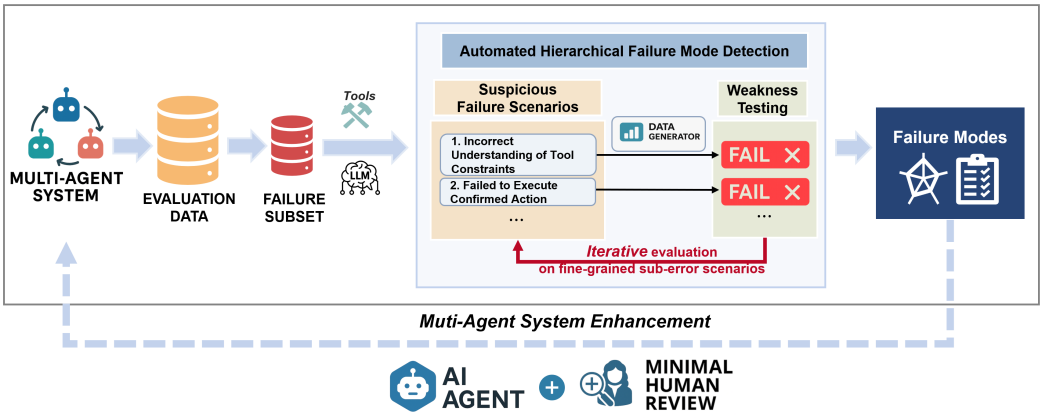


Figure 1: Overview of the automated hierarchical failure mode detection pipeline for agentic AI systems, transforming execution traces into validated failure modes that enable iterative system improvement with minimal human intervention.

We make the following contributions:

- A trace-driven framework for hierarchical failure mode detection in multi-agent systems.
- A statistical validation method for separating systematic failures from stochastic noise under limited data.
- An automated and human-guided mitigation loop for iterative system refinement.

The paper is organized as follows. Section 2 reviews related work; Section 5 presents the end-to-end detection, validation, and mitigation pipeline; Section 4 discusses hidden attribution biases and evaluation uncertainty; and Section 5 concludes the paper.

2 Related Work

Failure Attribution and Taxonomies Two complementary lines of work study MAS failures. Zhang et al. (2025c) frame failure attribution as a supervised task, finding that even the best method

achieves only 53.5% agent-identification accuracy and 14.2% step-localization accuracy on the Who&When benchmark, underscoring the difficulty of fine-grained attribution in multi-agent systems. Cemri et al. (2025) instead provide a taxonomy-driven retrospective analysis (MAST-Data: 1,600+ traces, 14 categories, κ=0.88), but focus on post-hoc classification rather than actionable diagnosis or iterative system improvement.

Positioning Both lines illuminate the hardness of MAS failure analysis, yet neither shows how identified failures can be systematically leveraged for system improvement. Our framework closes this gap by automatically detecting, statistically validating, and directly mitigating failure modes from execution traces in production-scale multi-agent systems. Unlike per-trace failure attribution (Zhang et al., 2025c;b; Deshpande et al., 2025) and inference-time agent self-evolution (Alzubi et al., 2026; Zhou et al., 2026; Zhong et al., 2024), we perform label-free, population-level failure-mode discovery that drives source-code-level mitigation; a detailed contrast is given in Appendix J.

3 Towards Production-Ready Agentic Systems

We formalize key failure concepts (Section 3.1), build a conversational MAS and generate evaluation data (Section 3.2), present hierarchical failure detection and mitigation (Sections 3.3–3.4), and discuss generalizability (Section 3.5).

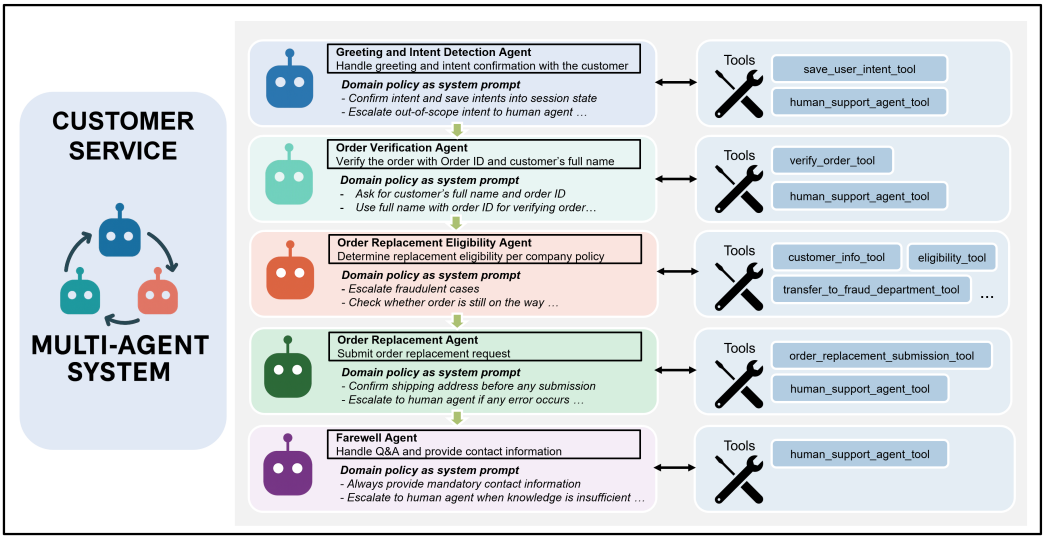


Figure 2: Overview of the customer service multi-agent system, where specialized agents handle order replacement steps and interact with external tools and human support.

3.1 Failure Definitions

We consider execution trajectories τ generated by a multi-agent system under a fixed evaluation protocol with binary outcome $Z(\tau) \in \{0, 1\}$, where $Z(\tau) = 0$ denotes failure.

An *execution error* is any agent action or interaction that violates task requirements under the evaluation protocol. A *decisive error* is an agent–time pair (i, t) (agent i at time t) in a failed trajectory τ such that intervening on the action at (i, t) yields a modified trajectory $\tau^{(i,t)}$ with $Z(\tau^{(i,t)}) = 1$ (success), following prior work (Zhang et al., 2025c).

An *observed failure scenario* is a recurring structured pattern of execution errors across failed trajectories.

A *failure mode* is an observed failure scenario whose associated trajectories exhibit a statistically significant lower success rate than a reference benchmark (e.g., a baseline population or target suc-

cess rate). A *decisive failure mode* is a failure mode that admits decisive-error interventions, such that correcting the pattern can flip outcomes from failure to success under the evaluation protocol.

3.2 Conversational Multi-Agent System and Dataset

3.2.1 Conversational Order Replacement MAS

We build a conversational multi-agent system for customer service order replacement using the Google ADK (Google, 2026) framework. As illustrated in Figure 2, the system follows a structured, workflow-driven design composed of specialized sub-agents responsible for intent detection, order verification, eligibility assessment, order replacement execution, and conversation closure. Agents interact through shared session state and external tools, including customer information retrieval, replacement submission, and escalation to human.

To support multi-turn conversations and enforce business constraints (e.g., prohibiting replacement before eligibility verification), agent execution is controlled through explicit state transitions and conditional agent/tool callbacks rather than default sequential execution. This design reflects common production practices, where reliability and policy compliance are prioritized over maximal autonomy.

3.2.2 Conversational Evaluation Data Simulation

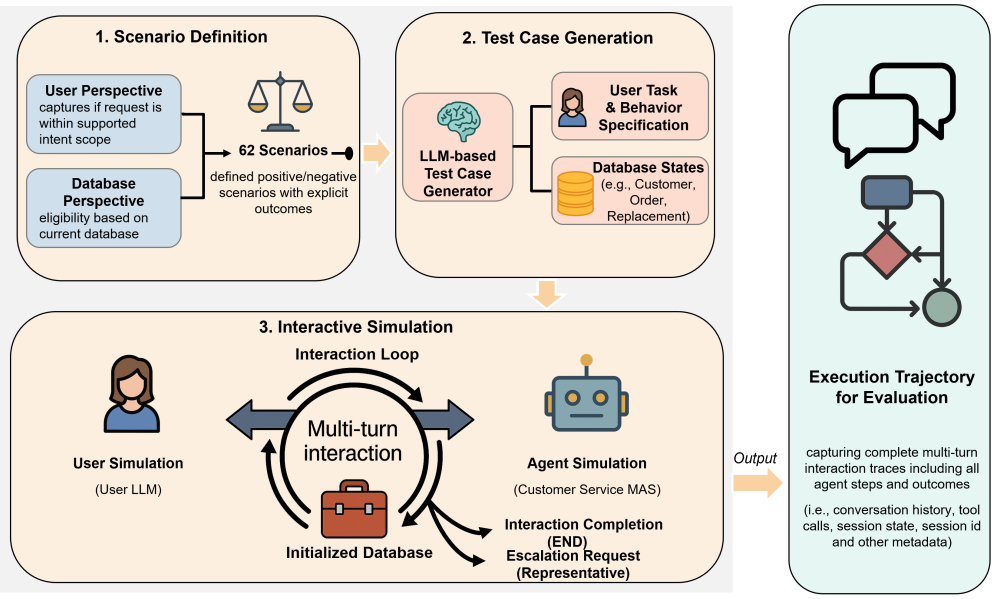


Figure 3: LLM-based conversational data simulation pipeline for generating multi-turn execution trajectories.

In the absence of historical interaction data, we evaluate the order replacement MAS using an LLM-based interactive user simulator that generates realistic multi-turn conversations with explicit expected outcomes. This approach enables large-scale, scenario-driven testing of agent behavior under realistic user interactions and dynamically generated backend states without manual data collection. Using this simulator, we generate 1,100 multi-turn interaction trajectories for evaluation.

As illustrated in Figure 3, simulated scenarios jointly specify user intent and backend conditions (e.g., order status, customer profile, and prior replacement history), with database states dynamically generated to satisfy business rules and policy constraints. Each interaction proceeds until task completion, escalation, or user termination, producing a full execution trace for downstream evaluation and failure analysis. Detailed scenario construction, simulator design, and example trajectories are provided in Appendix B.

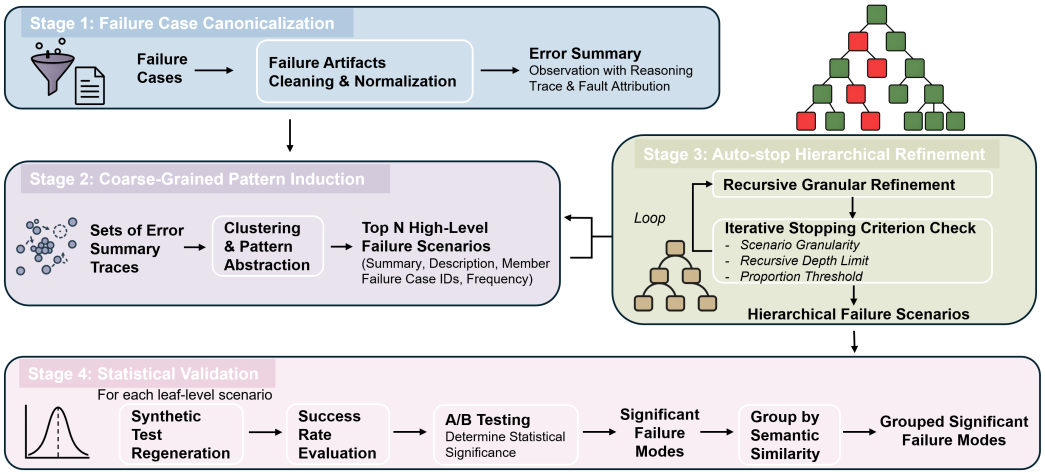


Figure 4: Four-stage pipeline for hierarchical discovery and validation of failure modes from execution traces.

3.2.3 Task Success Rate Evaluation

Task success is evaluated using a deterministic, rule-based criterion that compares observed system behavior against predefined expected outcomes. Specifically, the evaluation verifies (i) correct handling of replacement eligibility, (ii) appropriate execution of replacement actions, (iii) correct escalation to a human agent when required, and (iv) correct control flow decisions such as skipping inapplicable subagents as reflected in the session state. Under this protocol, the system achieves a task success rate of 74.9% on the 1,100 evaluation samples. We identify 276 failure cases, each corresponding to a failed execution trajectory (i.e., a data sample τ with $Z(\tau) = 0$). These failure cases form the failure subset used for detailed failure mode analysis in Section 3.3.

3.2.4 Model Configuration

Our system consists of three LLM-based components: (i) the conversational order replacement agent, (ii) the test case generator, and (iii) the LLM-based user simulator. Unless otherwise stated, we use *Gemini 2.5 Flash* (Team et al., 2024) as the default model for all components throughout the paper.

In the subsequent evaluation, we use *Gemini 3 Flash* as the LLM judge for large-scale data analysis due to its efficiency, and *Claude Opus 4.6* (Anthropic, 2026) for limited-prompt tasks where stronger reasoning is required despite higher overhead.

3.3 Automated Hierarchical Failure Mode Detection

We introduce an automated, hierarchical failure mode detection framework that leverages LLMs to identify, validate, and refine recurring failure patterns from large-scale, noisy execution data generated by the customer service multi-agent system. As shown in Figure 4, the framework consists of four structured stages, which we describe below.

Stage 1: Failure Case Canonicalization. Noisy, heterogeneous execution traces in MAS make failure pattern discovery challenging. To address this challenge, we employ an LLM judge (*Gemini 3 Flash*) to canonicalize each failure case. For every failed execution, the judge consumes the execution trace, session state, relevant database state, and agent instructions, and then distills this information into a concise error summary using a standardized prompt (Appendix C). Each summary consists of a clear error observation, structured reasoning describing the underlying cause, and a fault attribution flag indicating whether the failure originates from the agent itself or from non-agent factors (e.g., user disconnection, LLM API outages, or inconsistent backend state). This canonicalized representation, illustrated with an example in Appendix D, serves as the normalized input for subsequent failure pattern induction.

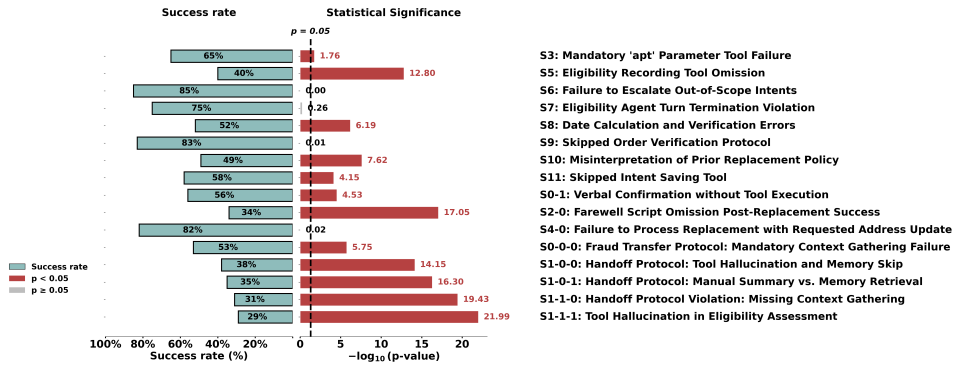


Figure 5: Success rates from large-scale synthetic testing ($n = 100$ samples per scenario) and corresponding statistical significance of leaf-level failure scenarios identified through hierarchical refinement.

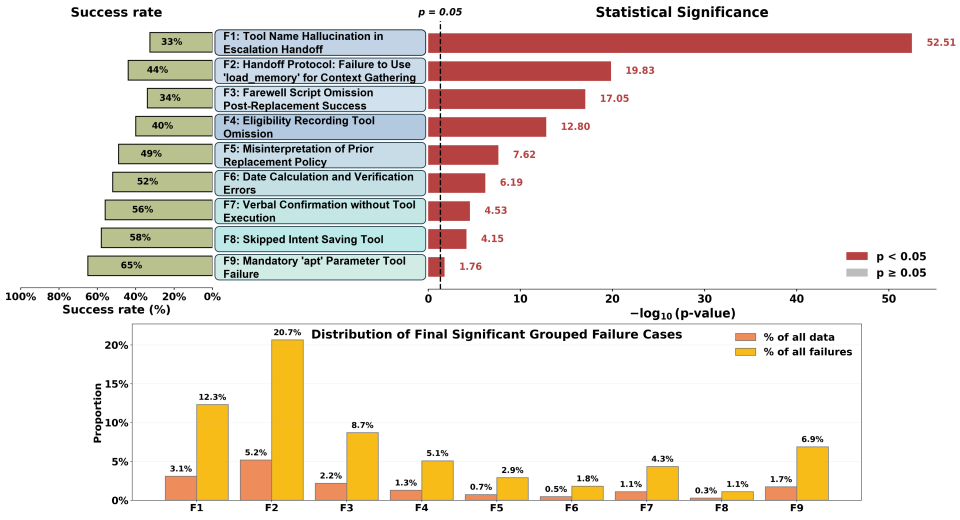


Figure 6: Top: Success rates from large-scale synthetic testing (left) and statistical significance as $-\log_{10}(p)$ (right) for grouped failure modes formed by clustering semantically similar significant leaf-level failure scenarios. The dashed line marks $p = 0.05$. Bottom: Proportion of each grouped failure mode among all data and all failures.

Stage 2: Coarse-Grained Failure Pattern Induction. Canonicalized failure summaries attributed to agent faults are aggregated and passed to the LLM judge (*Gemini 3 Flash*) for clustering and abstraction. In this stage, the judge induces the top $N = 12$ coarse-grained failure scenarios that capture the most frequently observed failure patterns in the dataset.

Each failure scenario is represented by a natural-language description, the associated set of failure case identifiers, and an empirical frequency based on the number of assigned cases. These coarse-grained scenarios provide an initial abstraction of potential systemic weaknesses for subsequent statistical validation.

Stage 3: Auto-stop Hierarchical Refinement. Coarse-grained failure scenarios provide high-level diagnostics but are often too broad to directly support actionable system improvements. To improve specificity, we recursively refine failure scenarios into more granular sub-scenarios. At each refinement step, the LLM judge (*Gemini 3 Flash*) is prompted once to propose semantically coherent sub-scenarios (see Appendix E for the full prompt specification). Refinement proceeds hierarchically and automatically terminates when a scenario becomes internally consistent, a predefined maximum depth is reached, or its estimated frequency falls below a minimum impact threshold.

Stage 4: Statistical Validation. To distinguish genuine system weaknesses from stochastic or incidental failures, we statistically validate only the leaf-level failure scenarios resulting from hierarchical refinement. For each leaf scenario, we regenerate $n = 100$ synthetic test cases under the same data generation conditions, measure the resulting task success rate p, and conduct a binomial hypothesis A/B test against a baseline success rate ($p_0 = 74.9\%$) with null hypothesis $H_0 : p \geq p_0$.

Leaf scenarios with p-values below the significance threshold ($\alpha = 0.05$) are deemed statistically significant failure modes, indicating systematic performance degradation. While the baseline p_0 can be flexibly defined to reflect business objectives, we adopt the current overall success rate of the multi-agent system as a reference to identify the most serious failure modes—those that contribute most substantially to overall performance decline. The discovered hierarchy is stable across runs, depths, and impact thresholds (Appendix K), and our conclusions are robust to multiple-hypothesis-testing correction via Bonferroni, Holm, and Benjamini–Hochberg (Appendix L).

Figure 5 summarizes the resulting observed leaf-level failure scenarios and their statistical significance. In the figure, scenarios S6, S7, S9, and S4-0 fail to reject the null hypothesis $H_0 : p \geq p_0$ with p-values > 0.05) and are therefore not statistically significant failure modes. After A/B testing, only statistically significant leaf-level failure scenarios are grouped. Highly similar significant leaf-level failure scenarios are consolidated into final grouped failure modes using a single prompt with *Claude Opus 4.6*, which reasons over the hierarchical context to merge closely related scenarios while preserving distinct and actionable failure patterns (see prompt details in Appendix F). Figure 6 presents the resulting failure modes along with their post-grouping success rates and statistical significance, all of which have p-value < 0.05.

3.4 Failure Mode–Guided Mitigation

We next leverage the statistically validated failure modes identified in Section 3.3 to guide targeted system mitigation.

Failure Mitigation Strategies Using the discovered failure modes as actionable signals, we evaluate two complementary mitigation strategies. In both cases, mitigation is applied to *grouped failure scenarios*, where individual failure scenarios are consolidated based on semantic similarity. We compare applying mitigation to groups derived from the full failure scenarios set versus those derived only from statistically significant leaf-level failure scenarios.

(1) Fully Automated Mitigation We implement a root-cause-driven code-fixing agent skill powered by *Claude Opus 4.6*. The agent operates on summarized failure mode descriptions produced by hierarchical detection and large-scale testing, without embedding direct access to raw execution logs. For each grouped failure scenario, the agent reasons over the observed failure patterns together with the source code of the customer service MAS and the `google-adk` framework, and applies targeted updates to the MAS codebase. All changes are recorded through detailed version change logs.

(2) Mitigation with Human Refinement In this setting, a single human developer reviews each grouped failure sceanrio using its descriptive summary and 1–3 representative failure samples, together with the automated agent's version change logs. Owing to the focused and abstracted presentation of grouped failure scenarios, the entire human review and subsequent code refinement across all scenarios typically requires less than one hour in total. The developer then applies targeted fixes to address missed root causes or strengthen partial corrections when necessary. The quantitative outcomes of these mitigation strategies are summarized in Figure 7 and Table 1. The following results summarize the primary empirical findings of failure mode–guided mitigation.

Table 1: Effect of Scenario Selection on MAS **Success Rates**

System Iteration	Full Failure Scenario Set (%)	Significant Leaf Failure Scenarios (%)
Baseline	74.91	74.91
Auto-Correction (Round 1)	78.91	**82.27**
Auto-Correction (Round 1) + Human	84.55	82.43
Auto-Correction (Round 2)	–	**85.27**

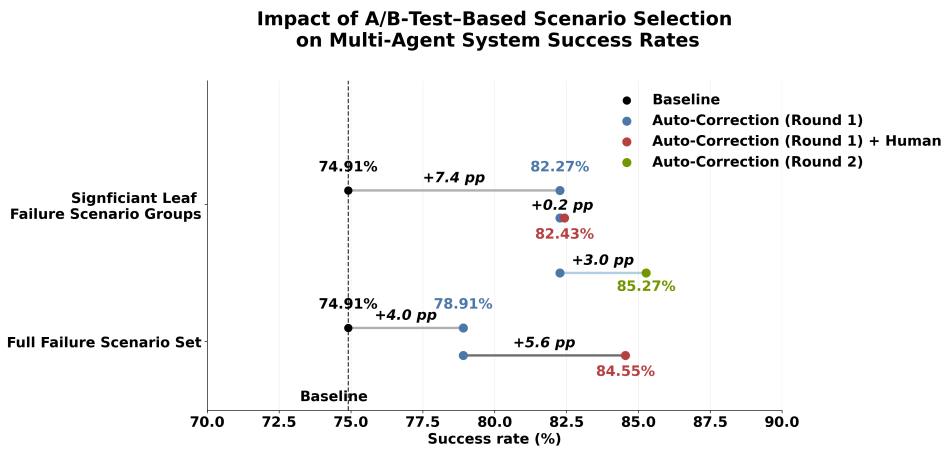


Figure 7: Effect of scenario selection on MAS success rates under automated and human-guided mitigation.

1. **Statistical significance testing focuses automation on high-impact failure modes.** Fully automated mitigation applied across all failure scenarios improves success from 74.9% to 78.9% ($+4.0$ pp). Restricting automation to statistically significant failure modes yields a larger improvement to 82.3% ($+7.4$ pp).
2. **Human refinement complements automation, with benefits depending on scenario scope.** When applied to the full failure set, human-guided refinement improves performance from 78.9% to 84.6% ($+5.6$ pp). Adding human refinement on top of already significant failure modes yields only marginal gains ($82.27\% \rightarrow 82.43\%, +0.2$ pp).
3. **Iterative automation delivers cumulative performance gains.** A second round of fully automated mitigation applied to statistically significant failure modes improves success from 82.3% to 85.3% ($+3.0$ pp), demonstrating support for iterative refinement.
4. **Focusing on fewer, high-confidence failure modes improves stability and efficiency.** Restricting automated mitigation to statistically significant failure modes yields larger and more reliable improvements than applying changes across all detected failures.

All identified failure modes were validated as correct and meaningful through human review; statistical validation, root-cause verification, and held-out evaluation further guard against incorrect failure descriptions propagating into faulty fixes (Appendix N). A single failure case may contain multiple execution errors, but only those that flip the outcome from success to failure are *decisive*. Accordingly, we distinguish *decisive failure modes*, which systematically induce failure under the evaluation metric, from *non-decisive failure modes*, which do not independently alter the final outcome. Two modes—F3 (*Farewell Script Omission Post-Replacement Success*) and F8 (*Skipped Intent Saving Tool*)—are labeled non-decisive, as they are not captured by the current metric. Nonetheless, both frequently co-occur with critical failures in practice, indicating that outcome-level metrics alone can overlook meaningful errors and motivating either metric refinement or dedicated monitoring of non-decisive failure modes.

Automated mitigation proves effective for localized failures such as prompt or tool-usage corrections, but less so when failures stem from deeper architectural or state-management issues–where human experts, leveraging system-level understanding, apply more comprehensive fixes. This delineates a practical boundary of automated mitigation in current agentic systems. Overall, the pipeline surfaces systematic weaknesses with minimal human effort: automated mitigation delivers scalable gains, while human intervention remains essential for complex, system-level failures, together providing a practical path to reliable production-grade multi-agent systems.

3.5 Generalization Beyond Multi-Agent Systems

While motivated by real world multi-agent systems, the proposed framework is not specific to MAS architectures. Because it treats execution traces as the primary unit of analysis, it naturally applies

to single-agent and multi-agent LLM systems with tool interaction. We evaluate the framework on multi-turn conversational single-agent workflows and benchmark-style agents on τ-bench (Yao et al., 2024), where it consistently identifies recurring, statistically significant failure modes that are not revealed by aggregate success rates alone. We further validate it on a dynamic-orchestration MAS (a DROP Q&A system with a free LLM orchestrator), where mitigation of discovered failure modes generalizes to held-out data (Appendix M). These results indicate that the methodology captures execution-level failure structure rather than MAS-specific artifacts. Full experimental details are reported in Appendix G.

4 Hidden Failure Attribution in Multi-Agent Systems

4.1 Blind Spots and Biases of LLM-as-a-Judge

Key insight: LLM-based judges can systematically overlook important failure modes due to shared assumptions with the evaluated system or low-frequency policy violations, leading to hidden reliability risks.

We observe that when prompts or instructions are underspecified, LLM judges may implicitly adopt the same interpretation as the agent under evaluation, causing critical failures to go unreported. In addition, rare but high-impact policy violations are often masked by more frequent yet less critical errors in large-scale data, making them unlikely to be surfaced through automated judgment alone. These limitations motivate combining LLM-based analysis with scenario-driven testing, deterministic checks for hard constraints, and statistical validation to reliably detect impactful failures. Detailed case studies and quantitative analyses are provided in Appendix H.

4.2 Diagnosing Uncertainty in Multi-Agent Systems

Key insight: Apparent performance of a fixed multi-agent system can vary substantially due to user simulation and scenario distribution shifts, even when the system itself remains unchanged.

We observe that replacing the LLM-based user simulator–while keeping the MAS backbone LLM fixed–can induce large changes in measured task success rates, highlighting significant simulator-induced uncertainty. This sensitivity cautions against interpreting offline evaluation metrics as absolute indicators of system quality without continuous monitoring and validation against production behavior. Beyond overall success rates, scenario-level analysis combined with statistical testing enables identification of genuinely underperforming business scenarios and supports principled prioritization for mitigation and future capability expansion. Detailed experimental results and analyses are provided in Appendix I.

5 Conclusion

We presented an end-to-end framework for detecting, validating, and mitigating failure modes in agentic AI systems operating under real-world constraints. Rather than relying on benchmark-centric evaluation or aggregate success metrics, our approach treats statistically validated failure modes as the primary abstraction for understanding and improving complex multi-agent behavior.

Through large-scale simulation, hierarchical failure abstraction, and targeted mitigation, we showed that focusing on high-confidence failure modes yields larger, more stable performance gains than indiscriminate correction. Our results further demonstrate that while automated mitigation scales well for localized failures, human intervention remains essential for addressing architectural and state-management issues that exceed the scope of automated reasoning. Importantly, this hybrid process requires minimal human effort and naturally supports iterative system refinement.

Overall, this work argues for a shift from benchmark-driven evaluation toward failure-mode–driven system maintenance for agentic AI. By grounding evaluation, diagnosis, and mitigation in statistically validated execution failures, the proposed framework offers a practical path to building and sustaining reliable agentic systems in production. We discuss the main limitations of our approach in Appendix O.

AI USE STATEMENT

In this work, generative AI tools were used as components of the experimental pipeline for automated failure discovery, failure canonicalization, and evaluation. Generative AI tools were also used for limited editing and language refinement of the manuscript. Generative AI tools were not used to generate the core scientific contributions, research claims, experimental conclusions, or reported results. All AI-assisted outputs used in the study were subject to automated validation procedures, aggregate result analysis, and author review where appropriate. Statistical analyses, experimental findings, methodological decisions, and final conclusions were independently reviewed and validated by the authors. The authors take full responsibility for the content of this paper, including all text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies methods for systematically identifying failure modes in agentic AI systems. The proposed framework is intended to improve robustness, reliability, and safety by exposing previously unknown failure behaviors before deployment. The experiments use publicly available benchmarks and commercial or publicly accessible LLM services and do not involve human subjects, personally identifiable information, or sensitive user data. We acknowledge that automated failure generation and evaluation may inherit biases present in underlying models or benchmarks, and therefore recommend human oversight and validation when applying the framework in practice.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, the paper provides detailed descriptions of the proposed methodology, evaluation pipeline, prompting framework, failure canonicalization procedure, clustering methodology, benchmark construction process, and statistical testing protocol. Experimental settings and implementation details are described in Section , while additional prompts, examples, dataset descriptions, and implementation details are provided in the appendix and supplementary material. The experiments are conducted using publicly available benchmarks and commercially available LLM APIs. These materials are intended to enable independent verification and replication of the reported findings.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Anthropic. Introducing Claude Sonnet 4.5, September 2025. URL https://www.anthropic.com/news/claude-sonnet-4-5. Accessed: 2026-05-06.

Anthropic. Claude Opus 4.6 System Card, February 2026. URL https://www-cdn.anthropic.com/0dd865075ad3132672ee0ab40b05a53f14cf5288.pdf. Accessed: 2026-05-06.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Derek Chen, Howard Chen, Yi Yang, Alexander Lin, and Zhou Yu. Action-based conversations dataset: A corpus for building more in-depth task-oriented dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3002–3017, 2021.

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

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

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL-HLT*, 2019.

Longling Geng and Edward Y Chang. Realm-bench: A real-world planning benchmark for llms and multi-agent systems. *arXiv preprint arXiv:2502.18836*, 2025.

Google. Google agent development kit (adk), 2026. URL `https://github.com/google/adk-python`. Accessed: May 2026.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, 2024.

Nikita Gupta, Riju Chatterjee, Lukas Haas, Connie Tao, Andrew Wang, Chang Liu, Hidekazu Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, et al. Deepsearchqa: Bridging the comprehensiveness gap for deep research agents. *arXiv preprint arXiv:2601.20975*, 2026.

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

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

Elad Levi and Ilan Kadar. Intellagent: A multi-agent framework for evaluating conversational ai systems. *arXiv preprint arXiv:2501.11067*, 2025.

Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, et al. Agencybench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. *arXiv preprint arXiv:2601.11044*, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Tie Ma, Yixi Chen, Vaastav Anand, Alessandro Cornacchia, Amândio R Faustino, Guanheng Liu, Shan Zhang, Hongbin Luo, Suhaib A Fahmy, Zafar A Qazi, et al. Maestro: Multi-agent evaluation suite for testing, reliability, and observability. *arXiv preprint arXiv:2601.00481*, 2026.

Sushant Mehta. Beyond accuracy: A multi-dimensional framework for evaluating enterprise agentic ai systems. *arXiv preprint arXiv:2511.14136*, 2025.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

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.

Dany Moshkovich, Hadar Mulian, Sergey Zeltyn, Natti Eder, Inna Skarbovsky, and Roy Abitbol. Beyond black-box benchmarking: Observability, analytics, and optimization of agentic systems. *arXiv preprint arXiv:2503.06745*, 2025.

Melissa Z Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, et al. Measuring agents in production. *arXiv preprint arXiv:2512.04123*, 2025.

Chiwan Park, Wonjun Jang, Daeryong Kim, Aelim Ahn, Kichang Yang, Woosung Hwang, Jihyeon Roh, Hyerin Park, Hyosun Wang, Min Seok Kim, et al. A practical approach for building production-grade conversational agents with workflow graphs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 1508–1519, 2025.

Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4922–4951, 2025.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: A family of highly capable multimodal models. arxiv 2023. *arXiv preprint arXiv:2312.11805*, 2024.

Harsh Vishwakarma, Ankush Agarwal, Ojas Patil, Chaitanya Devaguptapu, and Mahesh Chandran. Can llms help you at work? a sandbox for evaluating llm agents in enterprise environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9178–9212, 2025.

Wei Wang, Dan Zhang, Tao Feng, Boyan Wang, and Jie Tang. Battleagentbench: A benchmark for evaluating cooperation and competition capabilities of language models in multi-agent systems. *arXiv preprint arXiv:2408.15971*, 2024.

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 conversations. In *First conference on language modeling*, 2024.

Bingyu Yan, Zhibo Zhou, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, Zhoujun Li, Chaozhuo Li, and Xiaoming Zhang. Beyond self-talk: A communication-centric survey of llm-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.

Dehua Zhang, Qingsong Yuan, Lei Meng, Ruixue Xia, Wei Liu, and Chunbin Qin. Reinforcement learning for single-agent to multi-agent systems: from basic theory to industrial application progress, a survey. *Artificial Intelligence Review*, 2025a.

Guibin Zhang et al. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025b.

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. *arXiv preprint arXiv:2410.02644*, 2024.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025c.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, et al. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

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

Chenyang Zhu, Spencer Hong, Jingyu Wu, Kushal Chawla, Yuhui Tang, Youbing Yin, Nathan Wolfe, Erin Babinsky, and Daben Liu. Raffles: Reasoning-based attribution of faults for llm systems. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7659–7688, 2026.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Robert Tang, Heng Ji, et al. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025.

A Appendix

B Conversational Evaluation Data Simulation Details

After the first version of the order replacement multi-agent system is ready for evaluation, and in the absence of historical interaction data, we employ an interactive data simulator to generate evaluation data. Using this simulator, we collect 1,100 multi-turn interaction trajectories, each paired with an explicit expected outcome. The overall data simulation pipeline is illustrated in Figure 3.

Dataset generation proceeds in three steps.

(1) Scenario Definition with Expected Outcomes. We first define a diverse set of positive and negative scenarios grounded in business logic, requirements, and system scope, each with a predefined expected outcome—specifically, whether the agent should successfully complete the request or escalate it to a human representative. Scenarios are specified from two complementary perspectives: the *user perspective*, which captures whether the user request falls within or outside the supported intent scope, and the *database perspective*, which determines replacement eligibility based on customer profile, order status, and prior replacement history.

For example, a customer reporting a missing order is considered eligible for a replacement if the database shows that the order was delivered within the past 30 days, no prior replacement is currently in transit, and the customer profile indicates a low risk of fraud (i.e., fewer than ten returns in the last 30 days). In this case, the expected outcome is successful replacement issuance.

(2) Test Case Construction. Based on these factors, we define 2–3 positive and negative statuses for each database-related attribute (customer profile, order status, replacement history) and 3–5 statuses for each user-related attribute (e.g., request intent, address update). By combining these attributes and filtering non-applicable cases, we obtain 62 distinct scenarios with explicit expected outcomes. For each scenario, an LLM-based test case generator produces both (i) a user task and behavior specification and (ii) corresponding database states.

(3) Interactive Simulation and Trajectory Generation. The initialized database is exposed to the order replacement agent via tool interfaces, while the user task and behavior specification is provided to an LLM-based user simulator. The simulator engages in multi-turn interaction with the agent until termination or escalation. Each interaction yields a complete execution trajectory.

To ensure data realism and consistency, business constraints are enforced during generation. For example, delivery dates are dynamically generated within a 30-day window relative to the simulated conversation date, which is treated as the system's current date. Prior replacement identifiers are matching order and replacement records across databases.

An example execution trajectory is shown in Figure 8.

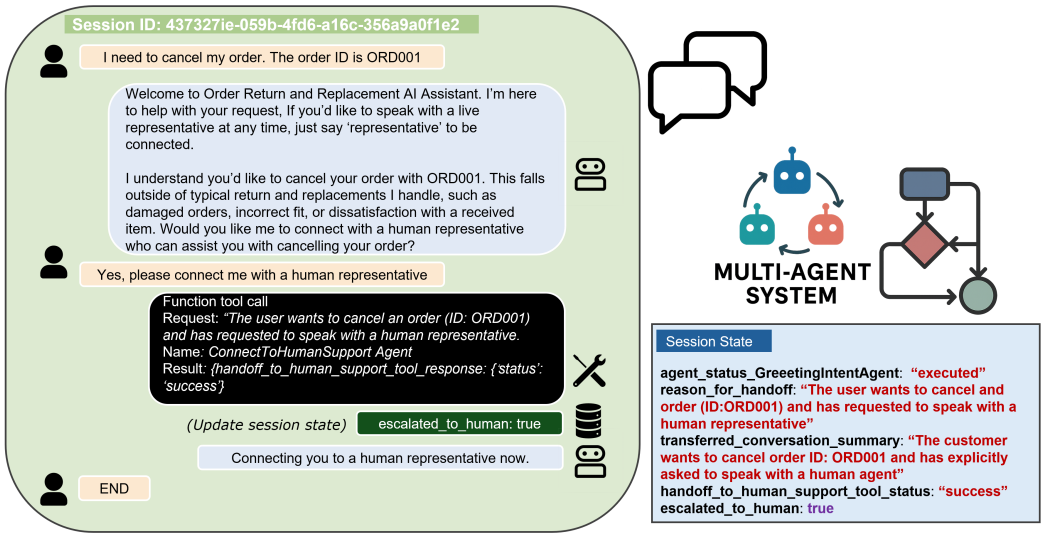


Figure 8: Example execution trajectory illustrating the user conversation, tool invocation triggering escalation, and resulting session state. Additional execution metadata are recorded but omitted for clarity.

C Prompt for Error Summary

This appendix presents the system prompt and task prompt used to generate error summaries. These prompts are applied to each raw execution trajectory from failure cases, together with critical associated metadata—such as system state, policy or prompt configurations, and database values—with the objective of identifying and extracting failure-related artifacts.

System Prompt

You are an expert AI Performance and Root Cause Analyst. Your mission is to dissect conversation history between a customer and an agentic AI to pinpoint the precise moment and reason for failure. You must produce a structured analysis with three specific outputs. You will be provided with:

1. **Policy / Prompt**: The AI agent's policy and the prompts it is expected to follow.
2. **Initial Database**: The customer's database profile prior to the conversation.
3. **Conversation Transcript**: The full transcript of the interaction between the customer and the agent.
4. **Final State**: The agent's final state after the conversation, including any important information that was stored.

Instructions:

1. Review the policy and prompts to understand what the agent is required to do and what it must not do.
2. Examine the customer's database profile to understand the initial background information.
3. Analyze the conversation transcript and compare it with the agent's final state to assess consistency and accuracy.
4. ⚠**CRITICAL – MANDATORY FULL TRAJECTORY AUDIT (REQUIRED BEFORE WRITING YOUR RESPONSE):**
   You MUST perform **two separate exhaustive checks** before producing your final structured response. Do NOT skip either check.

Check A — Policy Compliance (if Policy / Prompt is provided):
Go through **each and every instruction or rule** in the policy **one by one**, in order, and explicitly verify whether the agent adhered to it during the conversation. For each instruction:

- State the instruction.
- Determine whether the agent followed it, violated it, or partially followed it.
- If violated or partially followed, note the specific deviation with supporting evidence from the transcript.

Check B — Tool Call & Decision Audit (ALWAYS REQUIRED, regardless of policy availability):
Carefully examine every agent action and decision in the trajectory and flag ANY of the following:

- **Skipped tool calls**: Did the agent fail to call a tool it was supposed to call (e.g., skipped verification, skipped confirmation, skipped lookup)?
- **Incorrect tool calls**: Did the agent call the wrong tool, pass wrong arguments, or call a tool with stale/incorrect state information?
- **Failed tool calls**: Did a tool return an error or unexpected result, and did the agent handle it correctly?
- **Wrong decisions after tool results**: Did the agent misinterpret or ignore a tool's output and make an incorrect decision as a result?
- **Premature or missing actions**: Did the agent act too early (before gathering required info) or fail to act when it should have?

⚠Your `observation` and `reasoning` MUST reflect ALL failures found across BOTH checks — not just policy violations and not just tool issues. If both types of failures exist, the most impactful one should be the primary `observation`, but ALL findings must be captured in `reasoning`.

When analyzing the provided conversation, your response MUST be structured into these three distinct parts:

1. **Observation:** A single, highly-specific sentence that describes the **most critical** point of failure. This may be a policy violation, a skipped/incorrect/failed tool call, or a wrong decision — whichever had the greatest impact on the outcome. Do not just state *what* failed; state *how* it failed by including the specific data points, parameters, tool names, or actions involved.
   - **Vague Example (What to avoid):** "The AI failed to authenticate the user."
   - **Specific Example — Tool Call Failure (What to provide):** "The AI failed to authenticate the user because it incorrectly passed the zip code '94107' to the `authenticate_user` tool, even though the user had stated their zip code was '94102'."
   - **Specific Example — Skipped Tool Call (What to provide):** "The AI confirmed the order cancellation without ever calling the `cancel_order` tool, leaving the order unchanged in the system."
   - **Specific Example — Policy Violation (What to provide):** "The AI correctly used all user-provided information (Order #123, Zip 90210), but authentication failed, indicating the user's information was not found in the backend system."
2. **Reasoning:** A detailed root cause analysis that explains *why* the failure occurred. This must include:
   - **First, a category for the root cause:** `NLU Failure`, `State Tracking Error`, `Flawed Logic`, `Tool Execution Error`, `Tool Design/Arguments Error`, `Policy/Design Gap`, `User-Provided Error`, or `External System Failure` etc.
   - **Second, a brief explanation** for your choice.

- **Third, a direct quote** from the transcript as evidence to support your analysis.

3. **Fault Attribution:** A single boolean (`True` or `False`) that assigns responsibility for the failure.
   - Use `True` if the failure is the **AI agent's responsibility**, meaning the agent (or its system design — including prompts, tools, callbacks, and policies) failed to correctly handle the request or failed to escalate appropriately. This includes but is not limited to the following root causes:
     - `NLU Failure`
     - `State Tracking Error`
     - `Flawed Logic`
     - `Tool Execution Error`
     - `Tool Design/Arguments Error` (e.g., backend tool parameter mismatch)
     - `Policy/Design Gap` — the agent's policy, prompt, or tool design led to an incorrect outcome (e.g., missing guardrails)
     - **Hard rule:** If the failure is caused by any issue in the agent's own design or implementation — including prompts, policies, state handling, callbacks, or tool/interface contracts — set `fault_attribution=True` and use the most appropriate agent-fault category (for example, `Tool Design/Arguments Error`, `Tool Execution Error`, `Flawed Logic`, or `Policy/Design Gap`).
   - Use `False` **only** if the failure is completely **unrelated to the AI agent's design or behavior** — i.e., the agent handled the situation correctly given its design, and the failure was caused purely by factors outside the agent's control. This includes but is not limited to the following root causes:
     - `User-Provided Error` — the user provided incorrect, incomplete, or contradictory information (e.g., wrong order ID, wrong name, user hung up)
     - `External System Failure` — a system outside the agent's control failed (e.g., network timeout). `External System Failure` applies only to failures outside the agent's design and behavior (for example, upstream outages, timeouts, or unavailable dependencies).

Do not provide any other text or explanation outside of these three structured parts.

User Prompt

Please act as an AI Performance and Root Cause Analyst. Analyze the following conversation history and provide a response strictly in the three-part format of `observation`, `reasoning`, and `fault_attribution`.

1. **Policy / Prompt**: {*instruction_text*}
2. **Initial Database**: {*database*}
3. **Conversation Transcript**: {*trajectory*}
4. **Final State**: {*state*}

D FAILURE CASE CANONICALIZATION EXAMPLE

To illustrate the output of the failure case canonicalization step, we provide an example generated by the LLM judge.

Failure Case ID: 8
Error Observation: The agent failed to submit an order replacement by skipping the

required call to the `order_replacement_submission_tool` and instead incorrectly informed the user that the request had been processed.

Reasoning: This failure results from flawed control logic and an omitted tool invocation. According to `ORDER_REPLACEMENT_INSTRUCTION`, the agent must (i) confirm the mailing address and (ii) execute the submission tool to process the request. While the agent correctly verified the order and confirmed eligibility, it transitioned directly to a confirmation message without executing the submission step. Consequently, the final system state indicates that no replacement was recorded in the database.

Evidence: The agent stated, "I have processed your request to ship the replacement to your current address on file," but the execution trace contains no invocation of the `order_replacement_submission_tool`.

Fault Attribution: True.

E PROMPT FOR PROPOSING SUB-SCENARIOS WITHIN FAILURE SCENARIOS

This appendix presents the system prompt and task prompt used to propose fine-grained sub-scenarios within broader failure scenarios. The goal is to decompose high-level failure categories into more specific and actionable sub-cases that support detailed diagnosis and targeted remediation.

System Prompt

You are an expert AI error scenario analyst. You are given an error scenario, its description, and representative examples with index. Your objective is to determine whether the error scenario and its description are too general. If so, decompose the scenario into more granular, business-specific sub-error scenarios based on the type of action being confirmed but not executed (such as exchange, return, modify, cancel, update address).
For each sub-error scenario, provide:

- A concise description of the scenario (summarize the AI agent failure type, context, and relevant user behaviors or interactions).
- The indices of ALL samples that belong to this sub-scenario (not just a few representatives).
- An example observation and reasoning from the list that best illustrates this scenario.

Output no more than {N} sub-error scenarios. If more granular sub-scenarios are possible, select the {N} most relevant or frequent ones.

User Prompt

You are an expert AI error scenario analyst. Given the following error scenario, its description, and representative examples with their indices, your task is to determine whether this scenario is too general for actionable analysis and, if so, to break it down into distinct, more granular, business-relevant sub-error scenarios.
Instructions:

1. If you can identify **more than one** clearly actionable sub-error scenario (i.e., two or more), set `needs_granular_investigation` to True.
   - In this case, **return each of these sub-error scenarios as a separate dictionary** in a list.
   - **Do NOT return or restate the original scenario as a sub-scenario.**
2. If you cannot identify **multiple** actionable, distinct sub-error scenarios, set `needs_granular_investigation` to False.
   - In this case, **do not return any sub-error scenarios**.
   - Do not restate or modify the original scenario.

Output Format:
If multiple (more than one) subscenarios are found, output:

```
{
  "needs_granular_investigation": true,
  "error_scenarios": [
    {sub-error-scenario-dict-1},
    {sub-error-scenario-dict-2},
    ...
  ]
}
```

If not, output:

```
{
  "needs_granular_investigation": false,
  "error_scenarios": []
}
```

Each sub-error scenario dictionary must include:

- `"scenario"`: a concise, specific name for the sub-error scenario
- `"description"`: a brief, business-action-focused summary of why this error occurs
- `"example_index"`: A list of the indices of ALL samples that belong to this sub-scenario. Every index that belongs to the sub-scenario must be included.
- `"sample_observation_and_reasoning"`: a dictionary with keys `"index"`, `"observation"`, and `"reasoning"` showing a representative example

IMPORTANT: `example_index` must contain every index that belongs to the sub-scenario — not just a few.
Additional requirements:

- Return **no more than {N}** of the most significant or frequent sub-error scenarios.
- **Never repeat the original scenario or description as a sub-scenario.**
- If there is only one actionable scenario, do not split further.

Scenario: "{scenario}"
Description: "{description}"
Representative examples: {representative_examples}

F Prompt for Grouping Similar Failure Modes

This appendix describes the prompt used to automatically group highly similar failure modes identified from failure cases. Grouping related failure behaviors supports scalable analysis and more effective downstream remediation.

User Prompt

Read the scenario_description.json, group only very similar scenarios, output a json named by grouped_scenario_description.json to the same folder with unique "new_scenario_id", "scenario_id_set" for original scenario id grouped, "new_description" for combining description and deduplicate.

G Failure Analysis for Benchmark LLM Agents

This appendix reports detailed generalization results of the proposed automated failure mode detection framework on benchmark-style LLM agents evaluated using τ-bench. These experiments

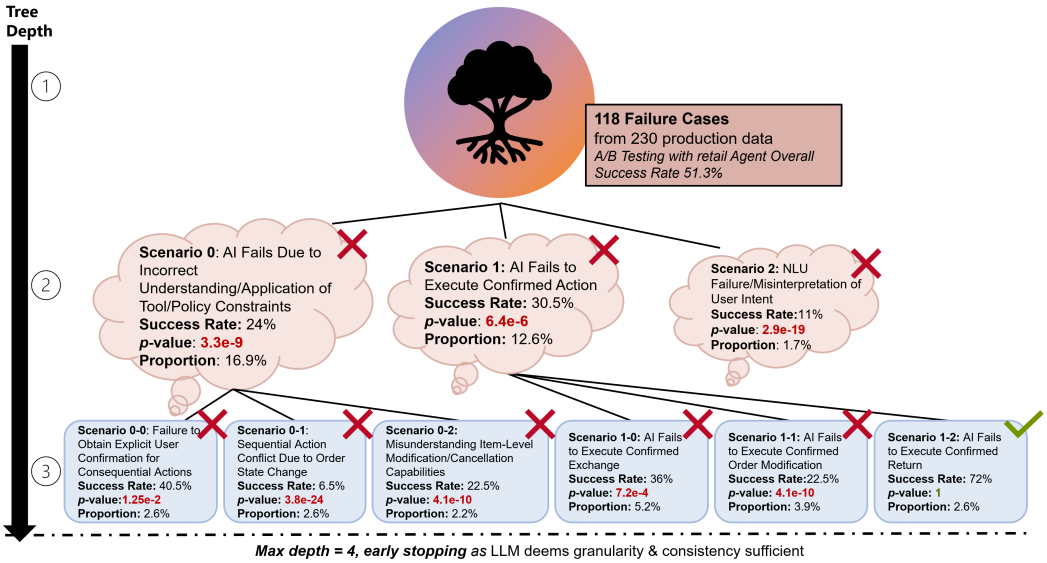


Figure 9: Hierarchical failure mode tree discovered by the proposed framework on the τ-bench retail domain. Failure cases are progressively clustered into coarse-grained scenarios and recursively refined into more specific sub-scenarios, with deeper levels indicating finer-grained failure modes.

complement the main results by demonstrating that the methodology is not limited to conversational multi-agent systems.

We evaluate the framework on both retail and airline tool-calling agents from the τ-bench benchmark. Using *Gemini 2.5 Flash*, we generate 230 retail test cases and 200 airline test cases, achieving task success rates of 51.3% and 40.5%, respectively. These outcomes yield approximately 100 failure cases per domain, providing sufficient data for meaningful failure mode analysis.

As illustrated in Figure 9, the hierarchical testing procedure progressively refines broad, first-layer error scenarios into increasingly granular and actionable sub-scenarios. Even when downstream A/B testing at deeper hierarchy levels produces a mixture of statistically significant and non-significant patterns, the framework amplifies sparse observations through large-scale data generation, enabling reliable isolation of genuine system weaknesses. Detailed results for all failure scenarios in hierarchical testing for both the retail and airline domains are reported below in Table 2 and Table 3.

To assess the quality and coverage of the discovered leaf-level significant failure modes, we employ *Claude Sonnet 4.5* (Anthropic, 2025) as an independent LLM judge to evaluate whether each failure case can be explained by at least one identified failure mode. The results show that 75.9% of retail failures and 90.8% of airline failures are covered by at least one failure mode, indicating that the derived failure modes provide high coverage of observed failure behaviors.

Finally, we note that additional performance improvements are likely attainable through more advanced prompt engineering of the instruction templates used by these tool-calling agents. Since the τ-bench agents evaluated here operate as single LLM agents, targeted prompt refinements represent a promising direction for future optimization.

Table 2: τ-bench **Retail** Failure Mode Tests: Statistical validation of failure scenarios. Success denotes large-scale testing success rate. A/B test p-values are reported in scientific notation; "True Failure Mode?" denotes statistically significant true failure modes ($p < 0.05$) or not.

Scenario ID	Failure Scenario	Description	Success Rate	A/B Test p-value	True Failure Mode?
0	AI Fails Due to Incorrect Understanding/Application of Tool/Policy Constraints	The AI's internal logic or interpretation of its own capabilities, tool requirements, or system policies is flawed. This leads to the AI making incorrect statements about what it can do, attempting actions that are impossible given current order states or tool limitations, failing to anticipate how one action impacts subsequent ones (e.g., order status changes), or violating explicit policy rules (e.g., single tool call, explicit confirmation requirement). User behaviors often involve requesting actions that challenge the AI's understanding of its operational boundaries or sequential processing.	24.0%	$3.30e{-09}$	Yes
1	AI Fails to Execute Confirmed Action	The AI successfully gathers all necessary information for a consequential action (e.g., exchange, return, modify order, update address) and receives explicit user confirmation to proceed, but then fails to call the appropriate tool to execute that action. This indicates a breakdown in the AI's workflow logic between confirmation and execution. User behavior typically involves providing a clear 'yes' or similar affirmative statement after the AI has presented the proposed action.	30.5%	$6.35e{-06}$	Yes
2	NLU Failure / Misinterpretation of User Intent	The AI fails to accurately interpret the user's natural language input, leading to a misunderstanding of their implicit requests, non-literal confirmations, or descriptive details. This can result in the AI asking for information it already has, misinterpreting terms, or failing to connect related pieces of information within the conversation. User behavior includes using natural, conversational language that the AI struggles to parse precisely.	11.0%	$2.89e{-19}$	Yes
0-0	Failure to Obtain Explicit User Confirmation for Consequential Actions	The AI fails to explicitly ask for or wait for user confirmation before executing a consequential action (e.g., return, modify, transfer, cancel), violating policy requiring explicit consent for database updates or significant interactions.	40.5%	$1.25e{-02}$	Yes
0-1	Sequential Action Conflict Due to Order State Change	The AI attempts to perform a second action (e.g., exchange, address update) on an order after a prior action (e.g., return, item modification) has changed the order's status, making the subsequent action impossible according to policy or tool constraints.	6.5%	$3.81e{-24}$	Yes
0-2	Misunderstanding Item-Level Modification/-Cancellation Capabilities	The AI incorrectly states its ability or inability to modify or cancel specific items within an order, often due to misinterpreting tool capabilities (e.g., removing items vs. exchanging them) or policy limitations for pending orders.	22.5%	$4.11e{-10}$	Yes
1-0	AI Fails to Execute Confirmed Exchange	The AI successfully gathers all necessary information for an order exchange and receives explicit user confirmation to proceed, but then fails to call the appropriate exchange tool (e.g., "exchange_delivered_order", "exchange_delivered_order_items") to execute the action.	36.0%	$7.18e{-04}$	Yes

Continued on next page

Scenario ID	Failure Scenario	Description	Success Rate	A/B Test p-value	True Failure Mode?
1-1	AI Fails to Execute Confirmed Order Modification	The AI successfully gathers all necessary information for modifying an order (e.g., items, payment, address) and receives explicit user confirmation, but then fails to call the appropriate modification tool (e.g., "modify_pending_order_items", "modify_order", "modify_order_payment", "modify_order_items", "modify_pending_order") to execute the action.	22.5%	$4.11e{-10}$	Yes
1-2	AI Fails to Execute Confirmed Return	The AI successfully gathers all necessary information for returning an order or item and receives explicit user confirmation, but then fails to call the appropriate return tool (e.g., "return_delivered_order", "return_order") to execute the action.	72.0%	$1.00e{+00}$	No

Table 3: τ-bench **Airline** Failure Mode Tests: Statistical validation of failure scenarios. Success denotes large-scale testing success rate. A/B test p-values are reported in scientific notation; "True Failure Mode?" denotes statistically significant true failure modes ($p < 0.05$) or not.

Scenario ID	Failure Scenario	Description	Success Rate	A/B Test p value	True Failure Mode?
0	Flawed Logic: Policy Violation or Failure to Act on Confirmed Intent	The AI agent frequently fails to correctly apply policy rules before executing a tool, leading to actions that violate constraints (e.g., modifying unmodifiable flights, using invalid payment methods, proceeding without mandatory information, or failing to verify user details). It also often fails to trigger the appropriate tool call even after explicit user confirmation and all necessary information has been gathered, indicating a breakdown in the transition from understanding to action.	25.5%	7.11e−04	Yes
1	Flawed Logic: Misinterpretation of Tool Output or Policy/Data	The AI agent misinterprets the output received from its tools or incorrectly applies policy rules, leading to incorrect information being conveyed to the user, false confirmations of actions, or flawed internal calculations. This includes situations where the AI confirms a change that the tool output indicates did not occur, misinterprets policy conditions for eligibility (e.g., cancellation, compensation), or makes incorrect calculations based on available data.	25.0%	4.79e−04	Yes
2	Conversational Flow & NLU Issues	The AI agent struggles with fundamental aspects of natural language understanding and conversational management. This includes failing to correctly parse user input, hallucinating information, getting stuck in repetitive loops, interrupting the user, or misinterpreting the user's intent or conversational state, leading to frustration and inefficiency.	10.5	2.93e−12	Yes
0-0	Failure to Execute Confirmed Action	The AI agent successfully gathers all necessary information and receives explicit user confirmation for an action (e.g., booking, cancellation, modification) but fails to trigger the appropriate tool call to execute the action. This indicates a breakdown in the AI's ability to transition from understanding and confirmation to execution.	19.5%	2.30e−06	Yes
0-1	Policy Violation - Incorrect Application of Business Rules	The AI agent attempts to execute a tool or takes an action that directly violates established business policies or constraints, such as modifying an unmodifiable flight type, changing prohibited reservation attributes (e.g., destination), or misapplying eligibility rules (e.g., refund policy). This indicates a flaw in the AI's understanding or application of policy logic.	33.0%	5.99e−02	No
0-2	Policy Violation - Insufficient Pre-execution Validation/Data Collection	The AI agent proceeds with a tool execution without first performing mandatory pre-execution checks, such as verifying user identity against reservation details, collecting all required information (e.g., reason for cancellation, explicit DOB), or adhering to specified data collection sequences. This leads to actions based on incomplete or unverified data, or actions that violate procedural policies.	29.5%	1.05e−02	Yes
1-0	AI confirms unexecuted updates based on flawed tool output interpretation	The AI agent incorrectly informs the user that a requested update or change (e.g., cabin class, flight details) has been successfully applied, despite the tool's output explicitly indicating that the change was not reflected or executed in the reservation details. This represents a failure to reconcile the intended action with the actual outcome reported by the tool.	31.0%	2.37e−02	Yes
1-1	AI misapplies policy rules for eligibility or action validity	The AI agent incorrectly interprets or applies business policy rules related to user eligibility for actions like compensation, cancellation, or valid payment methods, leading to incorrect denials, approvals, or attempts to use invalid options. This indicates a misunderstanding of the conditions or scope of a policy.	30.0%	1.40e−02	Yes

Continued on next page

Scenario ID	Failure Scenario	Description	Success	A/B Test p-value	True Failure Mode?
1-2	AI miscalculates or misextracts data from tool output	The AI agent incorrectly processes numerical data (e.g., total cost) or extracts specific data points (e.g., destination, flight details) from tool outputs, leading to inaccurate information being conveyed to the user or subsequent tool calls failing due to incorrect parameters. This highlights issues with data parsing, aggregation, or interpretation from structured tool responses.	21.0%	1.19e−05	Yes
2-0	NLU - Entity Extraction & Contextual Understanding	The AI fails to accurately extract specific entities (e.g., IDs, dates, preferences) from user input, hallucinates information, or misinterprets the user's intent due to an inability to correctly infer meaning from partial utterances or conversational context. This leads to the AI using incorrect data or misunderstanding the core request.	8.5%	5.02e−14	Yes
2-1	Conversational State Management & Repetitive Loops	The AI fails to maintain an accurate internal model of the conversation's progress and user's state. This results in the AI getting stuck in repetitive loops, re-asking for information already provided, failing to process explicit confirmations, or not adapting its dialogue strategy when the user indicates difficulty or provides specific commands (like 'STOP').	4.0%	8.50e−19	Yes
2–2	Conversational Turn-Taking & Interruption Issues	The AI struggles with managing the natural flow of dialogue, leading to premature interruptions of the user, repeatedly prompting the user to complete their sentence when they are still formulating a thought, or otherwise disrupting the natural back-and-forth of the conversation.	23.5%	1.34e−04	Yes
0-2-0	Cancellation Policy Violation: Missing Mandatory Data	The AI agent proceeds with a reservation cancellation without first obtaining all mandatory information (e.g., reason for cancellation) or adhering to the required data collection sequence, violating established policy.	24.5%	3.18e−04	Yes
0-2-1	Identity/Status Verification Failure for Critical Actions	The AI agent fails to verify the user's identity (e.g., user ID mismatch) or critical status (e.g., membership tier) against system records before proceeding with a sensitive action or providing policy-dependent information, leading to potential unauthorized actions or incorrect information.	84.5	1.00e+00	No
0-2-2	Booking/Modification Data & Payment Validation Failures	The AI agent fails to explicitly collect or validate critical passenger data (e.g., Date of Birth) or payment methods against policy requirements before confirming or executing booking or modification actions, leading to incorrect data or tool execution errors.	11.5	1.90e−11	Yes

H Blind Spots and Biases of LLM-as-a-Judge

In many large-scale GenAI automation settings, LLMs act as judges either explicitly (e.g., for annotation or evaluation) or implicitly (e.g., by reviewing execution traces and proposing suspicious failure scenarios). Such automation assumes that the LLM can reliably surface major and meaningful issues. However, when an LLM systematically overlooks certain classes of failure modes, those failures remain unreported and become *hidden failure attributions*.

This appendix presents detailed case studies illustrating blind spots and biases observed when using LLMs for automated weakness detection in agentic AI systems, along with mitigation strategies.

H.1 Missing or Unclear Instructions in Prompts

One common source of blind spots arises from missing or insufficiently precise instructions in prompts. In one example, the user intent "request correction due to wrong quantity received" was implicitly interpreted by the system as a request for replacement rather than an in-order correction. This distinction was not explicitly clarified in the greeting and intent-detection subagent prompt.

This failure mode was successfully identified through targeted business-scenario testing. Scenario-level monitoring, based on 40 samples for the "request correction due to wrong quantity received" intent, showed a success rate of 25% in the initial system, which remained low at 20% in the best-performing system after a second round of automated, failure-mode–driven system improvements. These results indicate a persistent, unresolved weakness that appears independent of LLM judgment.

H.2 Low-Frequency but High-Impact Policy Violations

In production deployments, agentic systems are often subject to strict policies whose violation can incur severe risk or financial loss, even if such violations occur infrequently. These low-frequency but high-impact failures are less likely to be surfaced by LLM-based automation due to their limited representation in noisy large-scale data.

To illustrate this limitation, we evaluate compliance with the following mandatory identity-verification policy:

> *"You MUST verify the customer's identity before any further processing. This step is mandatory and cannot be skipped. Request the customer's full first name, last name, and order ID. If order verification fails, immediately escalate by calling* `ConnectToHumanSupportAgent`*."*

We construct adversarial user interactions that stall identity verification and run 100 simulations. The agent achieves a success rate of only 10%, frequently continuing to request the same information despite explicit user refusal instead of escalating. This behavior would likely cause significant user frustration in production.

These observations motivate separating policy requirements into (i) *hard policies*, which must be enforced deterministically with full compliance, and (ii) *soft policies*, where high—but not absolute—adherence is acceptable and can be tested at scale using LLM-driven evaluation, and should be actively monitored in production.

I Diagnosing Uncertainty in Multi-Agent Systems

Multiple sources of uncertainty arise in multi-agent systems, ranging from data distribution shift to model-level behavioral variability. In production settings, user behavior and request distributions can evolve rapidly as systems are updated and exposed to a more diverse user base.

To study these effects, we evaluate the customer-service MAS under different LLM-based user simulators while holding the MAS backbone LLM fixed. As shown in Figure 10, replacing the user simulator from *Gemini 2.5 Flash* to *Gemini 3 Flash* results in a substantial drop in observed task success rate from 84.55% to 71.91%. This performance degradation occurs despite no changes to the MAS itself, highlighting significant simulator-induced uncertainty in offline evaluation.

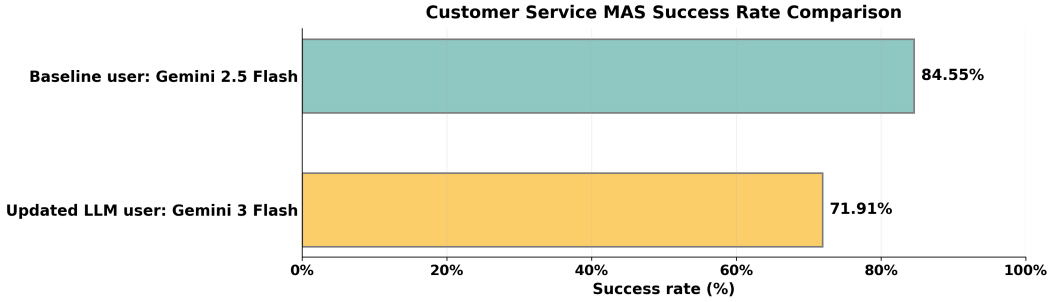


Figure 10: Customer service MAS success rates under different LLM-based user simulators, with the MAS backbone held constant.

These results underline the importance of continuous monitoring and scenario-level analysis when using synthetic or benchmark-driven evaluation to approximate production behavior. Changes in user simulation models can induce large apparent regressions, cautioning against interpreting offline success metrics as absolute indicators of system quality without accounting for simulator bias and uncertainty.

To address these limitations in a practical setting, we use τ-bench as a proxy for production data to demonstrate how business-scenario–level testing can be incorporated into production workflows to monitor and mitigate uncertainty arising from shifts in user behavior and request distributions. We treat the generated τ-bench evaluation data as proxy production data to illustrate how scenario-level analysis can further localize system weaknesses and characterize changes in user request distributions. Specifically, we extract and summarize business scenarios from the τ-bench retail domain and evaluate them through large-scale synthetic testing. Table 4 compares proxy production evaluation with automated large-scale testing results across key business scenarios. Several scenarios—including *Modify Pending Order (Item/Specs)*, *Cancel Pending Order*, and *Other*—exhibit statistically significant performance degradation (p-value < 0.05), indicating systematic weaknesses that are unlikely to arise from sampling noise alone.

ID	Business Scenario	Description	Production		Automated Large-Scale Testing		AB Test P-value
			Sample Size	Success Rate	Sample Size	Success Rate	
1	Exchange Delivered Item	The customer requests to exchange a product from a previously delivered order for a different variant, often due to size, color, features, or damage. This may involve a price difference and sometimes lead to a transfer to a human agent if the AI encounters limitations.	57	47.4%	200	45%	0.095980
2	**Modify Pending Order (Item/Specs)**	The customer requests to alter the specifications (e.g., color, size, material, features, upgrade/downgrade) of one or more items within an order that is still pending shipment. This often involves adjusting the price and refunding/charging the difference.	66	39.4%	200	34.5%	**0.000227**
3	Return Delivered Item	The customer requests to return one or more items from a delivered order for a refund, specifying the items, order details, and preferred refund method. This can be a partial or full return of an order.	53	56.6%	200	56%	0.834900
4	**Cancel Pending Order**	The customer requests to cancel a pending order, often stating reasons such as "no longer needed" or "ordered by mistake." This action typically applies to the entire order if partial cancellation is not supported by the AI.	17	64.7%	200	39.5%	**0.007134**
5	Information Inquiry	The customer requests specific details about an order or product, such as total price, item specifications, order status, or tracking numbers. This is a request for data retrieval rather than an action on an order.	14	57.1%	200	47.5%	0.215641
6	**Other**	Miscellaneous business scenarios not matching top types.	23	69.6%	200	43%	**0.042709**

Table 4: Comparison of Evaluation Results vs Automated Large Scale Testing Results Across Business Scenarios on τ-bench Retail Domain

Beyond weakness detection, this analysis enables principled monitoring of user request and business scenario distributions. By treating benchmark-derived scenarios as a proxy for production demand, shifts in scenario prevalence can be tracked over time, providing early signals of emerging unmet user needs. For instance, *Information Inquiry* requests account for approximately 7% of observed interactions in the evaluated dataset, suggesting that such queries represent a non-trivial user demand that may warrant consideration in future system iterations, if they are still outside the currently supported scope.

J Positioning: Failure Mode Discovery vs. Failure Attribution and Agent Self-Evolution

Our framework is easily conflated with two adjacent lines of work; we clarify the distinction here.

Failure attribution vs. failure mode discovery. *Failure attribution / issue localization* is a per-trace, typically supervised task: given *one* failed trace, predict *which agent* or *which step* is responsible, as targeted by Who&When (Zhang et al., 2025c), AgenTracer (Zhang et al., 2025b), and TRAIL (Deshpande et al., 2025). In contrast, *failure mode discovery* (this work) is a label-free, population-level task: given a corpus of executions with limited or no labels, automatically discover *recurring, statistically validated patterns of systematic weakness* that explain population-level performance degradation and drive iterative mitigation. Table 5 summarizes the difference.

Table 5: Per-trace failure attribution vs. population-level failure mode discovery.

Aspect	Failure attribution	Failure mode discovery (ours)
Input	One failed trace + ground-truth label	Corpus of unlabeled traces
Output	Failing agent + step index in that trace	Hierarchical, validated failure modes
Supervision	Requires labeled attributions	None; label-free discovery
Downstream use	Debugging one incident	Iterative system-level mitigation

These two tasks answer different questions ("who failed *here*?" vs. "what systematic weaknesses does this system have?") and produce different outputs. Per-trace attribution is therefore *complementary* rather than a competing baseline: its per-trace labels could serve as an additional input signal to our Stage 1 canonicalization, but cannot themselves produce statistically validated failure modes or guide mitigation. Indeed, Who&When (Zhang et al., 2025c) reports only 53.5% agent-identification and 14.2% step-localization accuracy, motivating why building on per-trace signals alone is fragile. For the actual task of failure-mode discovery, the natural comparison is LLM-as-a-Judge one-shot summarization and task-level success-rate evaluation, which we address in Section 4 and Appendix H.

Relation to agent self-evolution. Skill and memory evolution methods such as EvoSkill (**?**), Memento-skills (**?**), and MemoryBank (Zhong et al., 2024) modify an agent's behavior *at inference time* by adding skills, updating memory, or accumulating experience; the underlying agent code, prompts, and tool interfaces remain unchanged. Our mitigation instead modifies the *system source code itself*—prompts, tool signatures, callback logic, and control flow—guided by statistically validated failure descriptions, yielding a permanently corrected system version rather than learned runtime state. The two lines are complementary: self-evolution addresses adaptation to novel or evolving inputs, whereas our approach fixes systematic bugs and design gaps (e.g., tool hallucination in escalation handoff, policy misinterpretation) that no amount of runtime memory would repair without altering the underlying code.

K Stability of Hierarchical Failure Mode Discovery

Because hierarchical refinement relies on LLM-generated sub-scenarios, we assess how stable the discovered hierarchy is with respect to (a) independent runs / random seeds, (b) maximum refinement depth, and (c) minimum impact threshold. Using an LLM judge (*Gemini 3 Flash*) with a fixed similarity rubric (0–10), we score every pair of scenarios across two configurations and report bidirectional best-match coverage (fraction of scenarios in A whose best match in B has similarity $\geq 6/10$, and vice versa) together with average best-match similarity. Table 6 summarizes the results on the customer-service MAS.

Two independent runs under identical settings recover the same core failure modes (100% of the smaller set matched); the extra scenarios in run 2 are finer-grained refinements of already-covered patterns, not disjoint discoveries. Increasing depth progressively splits coarse scenarios into more granular sub-scenarios but never overturns the coarser layer, consistent with the auto-stop criterion in Section 3.3. Loosening the impact threshold from 4% to 2.5% adds only low-frequency scenarios while recovering the higher-impact ones exactly.

Table 6: Stability of discovered failure modes across configurations (customer-service MAS). Across every ablation, one direction achieves 100% coverage and average similarity stays between 8.6 and 9.3/10.

Ablation	# scenarios (A/B)	A covered by B	B covered by A	Avg. sim.
Judge run 1 vs run 2 (depth 3)	9 / 11	9/9 (100%)	9/11 (82%)	8.9 / 10
Depth 2 vs depth 3	4 / 9	4/4 (100%)	9/9 (100%)	8.6 / 10
Depth 3 vs depth 4	9 / 15	9/9 (100%)	14/15 (93%)	9.3 / 10
Threshold 2.5% vs 4%	9 / 6	8/9 (89%)	6/6 (100%)	9.1 / 10

Importantly, the three knobs (max depth, min-impact threshold, statistical test) are not independent tuning parameters but a joint safety mechanism against occasional LLM-judge hallucinations: single-trace patterns are filtered by the impact threshold, and whatever survives must reproduce as a *global* population-level effect under A/B testing (Stage 4). No single hyperparameter produces systematically different conclusions.

L Multiple Hypothesis Testing Correction

Because the leaf-level A/B tests constitute a family of hypotheses, we apply standard corrections to guard against false positives. We reran the $M = 16$ leaf-level tests against baseline $p_0 = 824/1100 \approx 0.749$ with Bonferroni (strict FWER), Holm (step-down FWER), and Benjamini–Hochberg (FDR at $q < 0.05$). Results are reported in Table 7.

Table 7: Raw and adjusted p-values for the $M = 16$ leaf-level failure scenarios ($n = 100$ per scenario, one-sided binomial, $H_0 : \pi_i \geq p_0$).

Scenario (leaf)	Succ.	p_{raw}	p_{Bonf}	p_{Holm}	Sig.?
S3 Mandatory 'apt' Param Tool Failure	0.65	1.73e-2	2.77e-1	8.67e-2	BH only
S5 Eligibility Recording Tool Omission	0.40	1.57e-13	2.52e-12	1.73e-12	All
S6 Failure to Escalate Out-of-Scope	0.85	9.95e-1	1.00	1.00	n.s.
S7 Eligibility Turn Termination Violation	0.75	5.47e-1	1.00	1.00	n.s.
S8 Date Calculation & Verification Errors	0.52	6.43e-7	1.03e-5	5.78e-6	All
S9 Skipped Order Verification Protocol	0.83	9.80e-1	1.00	1.00	n.s.
S10 Misinterpret. Prior Replacement Policy	0.49	2.42e-8	3.87e-7	2.42e-7	All
S11 Skipped Intent Saving Tool	0.58	7.03e-5	1.13e-3	4.22e-4	All
S1-0 Verbal Confirmation w/o Tool Execution	0.56	2.97e-5	4.75e-4	2.08e-4	All
S2-0 Farewell Script Omission	0.34	8.84e-18	1.41e-16	8.84e-17	All
S4-0 Failure to Process Address Update	0.82	9.64e-1	1.00	1.00	n.s.
S0-0-0 Fraud Transfer Context Gathering	0.53	1.78e-6	2.84e-5	1.42e-5	All
S1-0-0 Handoff Tool Hallucination & Memory Skip	0.38	7.13e-15	1.14e-13	7.13e-14	All
S1-0-1 Handoff Manual Summary vs. Memory	0.35	5.02e-17	8.03e-16	4.52e-16	All
S1-1-0 Handoff Missing Context Gathering	0.31	3.70e-20	5.92e-19	3.70e-19	All
S1-1-1 Tool Halluc. in Eligibility Assessment	0.29	1.02e-22	1.64e-21	1.02e-21	All

Of the 16 leaf scenarios, 12 are significant at raw $\alpha = 0.05$. Under the most conservative Bonferroni correction, 11 of these 12 remain significant; under Holm, the same 11 remain significant; and under Benjamini–Hochberg FDR control at $q < 0.05$, all 12 remain significant. The only scenario losing FWER significance is S3 (65% vs. 75%, $\Delta \approx 10$ pp), a borderline effect rather than a false discovery: it remains significant under BH-FDR and crosses Bonferroni/Holm in an independent 100-case regeneration. Crucially, the leaf scenarios that drive the mitigation experiments in Section 3.4 survive Bonferroni and Holm by many orders of magnitude, so the mitigation results and the paper's central claim—that focusing on statistically validated failure modes yields larger and more reliable gains than correcting all detected failures indiscriminately—are unchanged after correction.

M Generalization to Dynamic-Orchestration Multi-Agent Systems

The main-paper experiments cover deterministic-workflow MAS (customer service) and single-agent tool calling (τ-bench). To test the framework on a *dynamic-orchestration* architecture—where sub-agents are invoked based on a runtime orchestrator decision rather than deterministic chaining—we built an additional multi-agent reading-comprehension system on the DROP dataset (Dua et al., 2019) using Google ADK. A root LLM orchestrator dynamically routes each question to one of five specialized sub-agents (arithmetic, counting, sorting,

span-extraction, and ensemble), which may themselves invoke tools at their discretion. No deterministic chaining exists; the orchestrator decides routing at runtime.

Applying our framework to a discovery set of 800 DROP questions, we identified 5 statistically validated failure modes (Table 8). Notably, F1 (agent delegation / transfer-tool infrastructure) is unique to dynamic orchestration: it arises specifically because the LLM must decide routing at runtime, and without proper agent descriptions and transfer constraints the orchestrator attempts incorrect invocation methods. This failure mode would not appear in deterministically chained systems.

Table 8: Statistically validated failure modes discovered on the DROP dynamic-orchestration MAS.

ID	Failure Mode	Root Cause
F1	Agent Delegation & Tool Infrastructure Failure	Orchestrator fails to use `transfer_to_agent`; missing agent descriptions cause incorrect routing
F2	Output Formatting and Presentation Error	Agents append units, use number words instead of digits, or produce verbose spans
F3	NLU: Information Extraction & Entity Mapping	Agents confuse subject–object relationships or miss items during passage scanning
F4	Flawed Logic: Incorrect Mathematical/Comparative Reasoning	Wrong numbers selected, incorrect date calculations, conflicting data not reconciled
F5	State Tracking: Failure to Propagate Calculated State	Agent computes correct value in reasoning but returns a different/partial result

Based on the discovered failure modes, we applied targeted fixes (agent descriptions, transfer constraints, enhanced prompts with mandatory verification steps, improved tool return messages) and evaluated on both the 800-question discovery set and a held-out test set of 200 never-seen questions (Table 9). Improvements generalize to completely unseen data (+4.2 pp mean F1, +3.0 pp exact match, +4.5 pp F1$>$0.5 on the held-out set), confirming that the discovered failure modes reflect genuine systematic weaknesses rather than artifacts of the discovery distribution, with no regression on the discovery set. Together with the customer-service MAS (+10.4 pp) and τ-bench retail/airline (Appendix G), the framework is now validated across three architecture classes: deterministic-workflow MAS, dynamic-orchestrator MAS, and single-agent tool calling.

Table 9: DROP dynamic-orchestration MAS: mitigation results. Discovery-set averages over 3–4 runs; held-out test-set averages over 2 runs.

Dataset	Metric	Baseline	After Mitigation	Δ
Discovery set (800, seen)	Mean F1	0.892	0.892	—
Discovery set (800, seen)	F1 > 0.5	90.0%	90.3%	+0.3 pp
Held-out test (200, unseen)	Mean F1	0.882	0.924	**+4.2 pp**
Held-out test (200, unseen)	Exact Match	83.5%	86.5%	**+3.0 pp**
Held-out test (200, unseen)	F1 > 0.5	88.8%	93.3%	**+4.5 pp**

N Robustness of Mitigation to Failure Description Errors

A natural concern is that an incorrect failure description could propagate into a faulty fix. Our pipeline provides layered defenses. First, statistical validation (Stage 4) filters false alarms before any mitigation is applied: 4 of 17 leaf-level scenarios were correctly rejected as non-significant ($p > 0.05$), and restricting mitigation to validated modes yields $+7.4$ pp vs. $+4.0$ pp when applied indiscriminately (Table 1). Second, during automated mitigation the code-fixing agent independently validates the claimed root cause against the actual codebase rather than blindly trusting the description; in the DROP experiment (Appendix M) the agent found F1's description "partially accurate but surface-level" and recovered the true root cause through source-code analysis. Third, held-out evaluation on regenerated test data catches any faulty fix as performance degradation; no regression was observed in our experiments. Fourth, lightweight human review of grouped failure summaries and 1–3 representative samples per mode provides a final safeguard. Together, these mechanisms guard against incorrect descriptions propagating into system changes.

O Limitations

We consolidate the limitations discussed throughout the paper. (i) LLM-as-a-Judge blind spots: judges can adopt the same interpretation as the evaluated system and overlook rare but high-impact policy violations (Appendix H), motivating deterministic checks for hard constraints. (ii) Simulator-induced uncertainty: measured success rates can shift substantially under a different user simulator even with the MAS fixed (Appendix I),

cautioning against interpreting offline metrics as absolute. (iii) Boundary of automated mitigation: automation is effective for localized prompt/tool corrections but less so for deep architectural or state-management issues, where human experts remain essential (Section 3.4). (iv) Metric coverage: outcome-level metrics can miss non-decisive failure modes (e.g., F3, F8) that co-occur with critical failures, motivating metric refinement or dedicated monitoring.