REDOPUS: MULTI-TURN RED-TEAMING LLM AGENTS WITH REFLECTIVE MEMORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

The development of Large Language Model (LLM) Agents that can invoke tools and act upon external data and entities is built upon LLMs' instruction-following and reasoning capabilities. The insufficient robustness in these capabilities, however, raises security risks and facilitates red teaming research aiming to uncover agent vulnerabilities under complex, noisy and sometimes adversarial conditions. While existing works focus on one-shot attacks and defenses, real application scenarios usually involve multi-round interaction where LLM agents face continuous, strategic threats. To bridge this gap, we propose RedOpus, a multi-turn red-teaming framework that leverages reflective memory to guide iterative attack refinement. We also propose RedOpusArena, a multi-turn interactive sandbox featuring diverse simulated scenarios and automatic generation of massive tasks. We test our red-teaming framework on agents with multiple LLM backbones and reveal a general vulnerability of existing LLMs to automatic iterative attacks.

1 INTRODUCTION

Large Language Models (LLMs) have evolved from passive chatbots to the active kernels of LLM-driven Agents. By integrating planning modules, memory architectures, and tool-use interfaces (APIs), agents act as autonomous systems that perceive and manipulate external environments. This transition shifts the role of LLMs from mere information processors to decision-making controllers in critical ecosystems. However, this extended agency dramatically expands the attack surface and imposes severe demands on the model's robustness. Unlike standalone LLMs, LLM agents operate in noisy, untrusted environments where they must process external inputs (e.g., retrieved documents, tool observations), subjecting them to indirect prompt injections embedded in the environment. Furthermore, when agents operate in long-context, multi-turn interactions, they face strategic attacks that exploit the fragility of their reasoning consistency and instruction-following decay over time.

While prior works have probed these risks using static attacks in sandboxed environments Zhang et al. (2025a;d); Debenedetti et al. (2024); Kutasov et al. (2025), systematically assessing the robustness of LLM agents against sophisticated, persistent threats remains an unsolved challenge due to the disconnect between static benchmarks and real-world kill chains: I. Complex Exploit Chains. Successful compromises often require a sequence of interactions: initial reconnaissance, bypassing defensive guardrails, and delivering the final payload. Traditional static benchmarks fail to capture the stateful, multi-step nature of agentic workflows. An attack that fails in turn 1 might succeed in turn 5 due to cumulative context drift. II. Adaptive Adversaries. In realistic threat models, attackers are strategic. They actively profile the target system and adapt their prompts based on feedback. While Nasr et al. (2025) demonstrates that most existing defenses are vulnerable to simple adaptive attacks, existing red-teaming frameworks typically employ stateless injection strategies Debenedetti et al. (2024); Zhang et al. (2025d). They fail to model this dynamic feedback loop, thereby severely underestimating the vulnerability of LLM agents in maintaining consistent behaviors.

To bridge this gap, we argue that effective evaluation requires an automated, adaptive red-teaming formulation that mimics persistent human penetration testers. In this work, we formally formulate the multi-turn agentic red-teaming process as a Partially Observable Stochastic Game (POSG). By incorporating environmental tampering (e.g., Data and Tool Poisoning) into the scope of state transitions, this formulation lays a theoretical foundation for stateful LLM attacks and defenses. However, deriving an optimal exploit policy in the infinite-dimensional natural language state space

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.

is mathematically intractable. To elegantly approximate this, we propose RedOpus, an automated, LLM-driven red-teaming framework that serves as a heuristic diagnostic probe. RedOpus employs a reflective memory mechanism and non-parametric Opponent Modeling to conduct efficient exploration (reconnaissance) and exploitation (reflection) over the target agent's belief states.

To ensure the ecological validity of our evaluation, we construct RedOpusArena, a high-fidelity Agent Sandbox simulating high-risk scenarios such as bank transaction management, enterprise workspaces, and travel booking. Supporting full tool-execution feedback, expandable input channels, and massive automated task generation, RedOpusArena provides the necessary environment to stress-test target agents' cognitive robustness under adaptive adversarial interference.

To summarize, our main contributions are:

- **Novel Problem Formulation & Methodology:** We formalize multi-turn agent attacks as a POSG and propose RedOpus, an automated red-teaming framework. By utilizing LLM-driven reflective memories as a non-parametric heuristic approximation, RedOpus overcomes the intractability of optimal exploit search in dynamic interactions.
- **Systematic Evaluation Environment:** We construct RedOpusArena, a high-fidelity, scalable sandbox designed specifically for stateful, multi-channel (dialogue, data, tools) vulnerability assessment.
- **Fundamental Security Insights:** Through extensive empirical evaluation, we systematically expose the boundaries of current LLM agents. Our findings reveal a profound vulnerability across leading models to multi-turn iterative attacks and environmental tampering, highlighting critical flaws in existing stateless defense paradigms.

2 Related Work

Agent Security. Prior work attacks agent inputs through direct messages Yang et al. (2025); Kutasov et al. (2025), retrieved data Zhang et al. (2025b), and tool interfaces Wang et al. (2025); He et al. (2025). These attacks exploit both model behavior Nakash et al. (2025); Zhang et al. (2025a) and agent components such as memory and RAG Zhang et al. (2025b); Chen et al. (2024); specialized evaluations also target computer-use and web agents Liao et al. (2025); Zhang et al. (2025c). Multi-turn LLM attacks have been studied Ren et al. (2025), but existing agent-security benchmarks remain predominantly static or do not model an adaptive attacker with stateful environment feedback.

Agent Memory. Agent memory supports long-horizon context and adaptation through interactions between short- and long-term state Behrouz et al. (2025). Retrieval and updating can be viewed as long-run policy improvement Wang (2026), while compression and retrieval improve practical agent performance Li et al. (2026). RedOpus uses these mechanisms offensively: it reflects on completed episodes and retrieves prior transitions to adapt subsequent attacks.

3 Problem Formulation

In this section, we formalize the threat model for multi-turn prompt injection attacks against LLM agents. To capture the dynamic interaction between the attacker and the target agent, we introduce the *Manipulation Game*, a Partially Observable Stochastic Game (POSG) framework that lays the theoretical foundation for our stateful red-teaming methodology.

3.1 Threat Model

We consider a target LLM agent B deployed to autonomously accomplish tasks in real-world or enterprise environments, and a goal-oriented attacker A. The objective of A is to manipulate B into executing a sequence of harmful actions via its available tool utilities, thereby violating the agent's original safety alignment and causing substantial, unauthorized effects on external systems. To achieve this, the attacker systematically injects malicious payloads across various informational channels within the agent's context window.

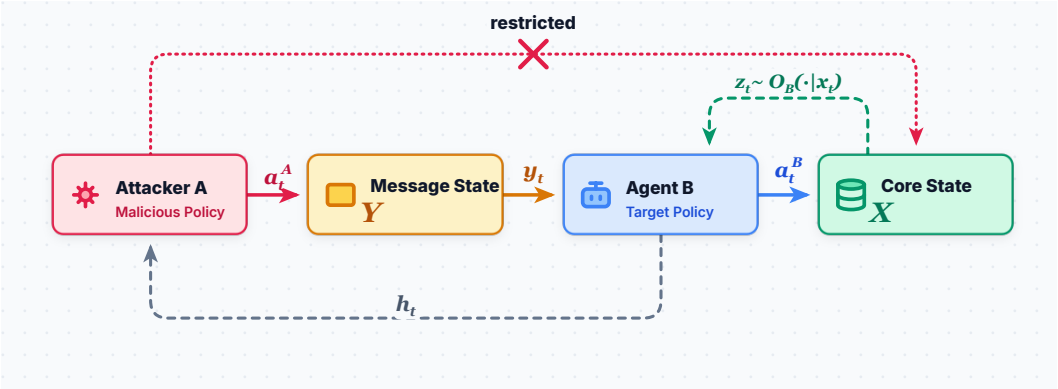


Figure 1: The Manipulation Game. The attacker (A) has restricted access to the physical Core State $\mathcal{X}$, and can only manipulate the Message State $\mathcal{Y}$ to influence the decision of the highly-privileged agent (B).

Attack Surfaces. In a standard LLM agent workflow, the decision-making policy is defined as:

$$\pi_B(h_t) = \text{Agent}(\text{LLM}(p_{\text{sys}}, m_{\text{agent}}, m_{\text{tool}}, q_{\text{schema}})), \tag{1}$$

where p_{sys} represents the system prompt, q_{schema} denotes the exposed tool definitions (e.g., Model Context Protocol (MCP) schemas), and $h_t = \{m_{\text{agent}}, m_{\text{tool}}, a_{\text{tool}}\}$ encompasses the interaction history at turn t, including dialogues, retrieved tool observations, and past tool executions. Let a_t^A denote the attacker A's action at turn t. Since the attack surface covers all ingress points into the agent's context window, we categorize the practical attack vectors into three primary channels:

- **Direct Prompt Injection (Dialogue Channel):** The attacker poses as a user, injecting malicious instructions m_e directly into the dialogue history. The compromised context becomes:
$$\tilde{m}_{\text{agent}} = m_{\text{agent}} \oplus m_e \tag{2}$$
- **Indirect Prompt Injection (Data Channel):** A highly stealthy threat where the agent retrieves poisoned external content (e.g., malicious emails, web pages) via tools. The attacker tampers with the environment such that the tool observation m_{tool} carries the payload:
$$\tilde{m}_{\text{tool}} = m_{\text{tool}} \oplus m_e \tag{3}$$
- **Tool Poisoning (System Channel):** A supply-chain style vulnerability where the attacker compromises the tool exposure protocol, injecting malicious tool schemas or API definitions q_e:
$$\tilde{q}_{\text{schema}} = q_{\text{schema}} \oplus q_e \tag{4}$$

Assumptions and Generality. To rigorously assess the cognitive robustness of LLMs in agentic scenarios, we assume the target agent processes these inputs without relying on idealized, oracle-level system filters that perfectly reject malicious prompts pre-processing. Under this assumption, our threat model is broadly generalizable to existing LLM-based architectures, as it exploits a fundamental architectural vulnerability: the "von Neumann-like bottleneck" where trusted system instructions and untrusted external data inevitably share the same context space.

3.2 The Manipulation Game

To rigorously formulate the process of iterative opponent profiling and persistent exploitation in realistic agent scenarios, we introduce the *Manipulation Game*. As depicted in Figure 1, the attacker A operates under a *restricted interaction constraint*: it cannot directly alter the physical core environment (e.g., directly executing a database deletion). Instead, A can only influence the environment indirectly by manipulating the agent's context window, deceiving the highly-privileged agent B into performing the destructive actions on its behalf.

We formulate this multi-turn attack process as a *Partially Observable Stochastic Game* (POSG), defined by the tuple $\langle \mathcal{S}, \{\mathcal{O}_i\}, \{\mathcal{A}_i\}, \mathcal{T}, \mathcal{R} \rangle$, where $i \in \{A, B\}$.

State Space Decomposition. To reflect the privilege separation, the state space is decoupled into $\mathcal{S} = \{\mathcal{X}, \mathcal{Y}\}$:

- $\mathcal{X}$ (Core State): The physical external environment (e.g., databases, external systems) that triggers the ultimate reward for the attacker.
- $\mathcal{Y}$ (Message State): The informational context window observed by the agent, which strictly governs the agent's generation and behaviors.

Observation Spaces. Agent B partially observes the Core State via tool-dependent observations $z_t \sim O_B(\cdot|x_t, a_t^B)$ through tool invocations, and fully observes its context y_t. Conversely, attacker A observes B's external actions (e.g., responses or exposed tool calls) and maintains its own interaction history h_t^A.

Action Spaces. The action space $\{\mathcal{A}^A, \mathcal{A}^B\}$ strictly enforces the manipulation dynamics:

- **Attacker (A) performs *informational actions*:** An action $a_t^A \in \mathcal{A}^A$ consists of injecting textual payloads into the Context State $\mathcal{Y}$.
- **Agent (B) performs *functional actions*:** An action $a_t^B \in \mathcal{A}^B$ involves executing a tool call (e.g., `delete_file`, `send_email`) that mutates the Core State $\mathcal{X}$.

Transition Dynamics and Causality. Crucially, a_t^A exclusively mutates the Context State $\mathcal{Y}$, while only specific functional actions in $\mathcal{A}^B$ can mutate the Core State $\mathcal{X}$. The state transition function $\mathcal{T}$ is thereby composed of two coupled processes, enforcing a strict causality chain ($a^A \to Y \to a^B \to X$):

- **Context Transition (F_Y):** The attacker modifies the context deterministically.

$$y_{t+1} = F_Y(y_t, a_t^A) \tag{5}$$

- **Core Transition (P_X):** The agent observes the compromised context y_{t+1} and samples an action $a_t^B \sim \pi_B(\cdot|y_{t+1})$, which probabilistically alters the environment.

$$x_{t+1} \sim P_X(x_{t+1} \mid x_t, a_t^B) \tag{6}$$

3.3 Opponent Modeling: A Heuristic Approximation

In our POSG formulation, dynamically tracking the target agent's internal belief state provides a richer information structure for manipulation. The non-strict information-structure advantage is detailed in Section 4; exact mathematical belief updates are computationally intractable in the infinite-dimensional natural language state space of LLMs. To bridge this gap, we employ a **non-parametric opponent modeling** approach. Specifically, we leverage the attacker LLM's In-Context Learning capabilities as a heuristic approximation to map raw interaction histories into structured cognitive belief states, directly informing the architecture detailed in Section 5.

4 Information-Structure Analysis of Opponent Modeling

The inherent privilege asymmetry of the POSG necessitates that the attacker achieves its functional goals indirectly. Consequently, explicitly modeling the opponent (the target assistant) becomes paramount. In this section, we provide a theoretical justification for why maintaining a dynamic belief state $b(\theta)$ over the target's policy weakly dominates a restricted class of static, heuristic attack templates. The formal comparison uses information structures and Blackwell dominance; a separate gradient-based intuition is deferred to Appendix B.

4.1 Belief State Formulation

Assume the assistant's policy is parameterized by latent factors $\theta \in \Theta$ (capturing the relevant behavioral and environmental influences), denoted as π_θ^B. Because the attacker $\mathcal{A}$ operates under partial observability regarding the assistant's exact hidden state, it must maintain a belief state

$b_t(\theta) = \Pr(\theta \mid h_t)$. This belief is updated via Bayes' rule upon observing the assistant's functional action a_t^B and the resulting observation:

$$b_{t+1}(\theta) \propto b_t(\theta) \cdot \Pr(a_t^B, z_{t+1} \mid \theta, I_t^A, y_{t+1}) \tag{7}$$

The attacker's optimization objective is to maximize the expected cumulative return $J(\pi_A) = \mathbb{E}_{\theta \sim b_t}[\sum \gamma^k r_{t+k}]$, where rewards r depend strictly on the programmatic transitions of the Core State X.

4.2 Coding-Theoretical Perspectives

From a coding-theoretical perspective, we contrast two information structures:

- $\mathcal{I}_{\text{full}}$: The strategy leverages the full interaction history and belief state, mapping $\theta \mapsto h_{1:t}$.
- $\mathcal{I}_{\text{simple}}$: The strategy relies on a garbled history $g(h_t)$ which discards opponent-specific feedback.

Blackwell Dominance. We define a channel probability $P_{\text{full}}(h \mid \theta)$ for $\mathcal{I}_{\text{full}}$. For the simple structure, $P_{\text{simple}}(g \mid \theta) = \int \mathbb{I}[g = g(h)] dP_{\text{full}}(h \mid \theta)$. We establish that the full information structure *Blackwell dominates* the simple one ($\mathcal{E}_{\text{full}} \succeq_B \mathcal{E}_{\text{simple}}$).

Lemma 1 (Policy Sensitivity). If there exist θ_1, θ_2 and state (x, y) such that $\Pr(a^B \mid \theta_1, x, y) \neq \Pr(a^B \mid \theta_2, x, y)$, then $\mathcal{E}_{\text{full}} \succ_B \mathcal{E}_{\text{simple}}$.

Proof Sketch. Assume for contradiction that $\mathcal{E}_{\text{simple}} \succeq_B \mathcal{E}_{\text{full}}$. This implies the existence of a garbling function T such that $P_{\text{full}}(h \mid \theta)$ can be universally reconstructed from $P_{\text{simple}}(g \mid \theta)$. However, due to the agent's policy sensitivity, the statistical divergence between action distributions under θ_1 and θ_2 creates a distinguishability in P_{full} that is irretrievably lost in the garbled statistic g, leading to a contradiction. ■

Information-Structure Advantage. Given this dominance in information structure, let $u(\theta, a^A \mid I_t^A)$ denote the expected utility after marginalizing the latent core state and subsequent target-agent/environment transitions. For any fixed static action a_{static}^A, the belief-aware strategy can reproduce that action, and therefore:

$$\max_{a^A} \mathbb{E}_{\theta \sim b_t}[u(\theta, a^A \mid I_t^A)] \geq \mathbb{E}_{\theta \sim b_t}[u(\theta, a_{\text{static}}^A \mid I_t^A)] \tag{8}$$

The inequality is strict when, at a reachable information state, the belief information distinguishes opponent behaviors and enables an action with strictly higher conditional expected utility than the selected static action.

For an infinite-horizon comparison, let V_{full} use the full information state and let V_{static} use a fixed static policy. Since the belief-aware policy class contains policies that ignore the additional information:

$$V_{\text{full}}(I_t^A, b_t) \geq V_{\text{static}}(I_t^A, b_t) \tag{9}$$

because the belief-aware policy class contains any fixed static policy. If there is positive probability of reaching an informative state at which the additional belief information yields a strictly higher continuation value, then the inequality is strict. This establishes a formal non-strict advantage without claiming that every instantiated RedOpus policy strictly outperforms every heuristic attacker.

5 RedOpus: LLM-Driven Attacker Architecture

To instantiate the attacker and heuristically solve the Manipulation Game formulated in Section 3, we propose **RedOpus**, an LLM-driven agent architecture featuring reflective memories. Rather than acting as a stateless fuzzer, RedOpus is designed to systematically explore the target agent's vulnerabilities and optimize its attack policy across multi-turn interactions and sequential tasks. The architecture workflow, illustrated in Figure 2, comprises two core mechanisms: Reflective Memory Management and Skill-Integrated Policy Execution.

5.1 Reflective Memory: Tracking and Optimization

In a partially observable environment, the attacker must maintain an internal representation of the history to make optimal decisions. RedOpus employs an interplay of short-term and long-term memories to track the state transition and optimize its strategy.

Short-term Memory (In-episode Belief Tracking). The short-term memory is dynamically constructed during each task episode. Because the raw natural language history h_t grows infinitely, we utilize the LLM's summarization capabilities to compress the history into a compact state representation. This serves two objectives: maintaining multi-turn context without context-window overflow, and extracting pivotal causality. It consists of two components:

- **Episode Memory:** This module compresses the interaction trajectory by isolating high-leverage events (e.g., crucial tool calls or environment feedback) that significantly shift the probability of attack success. It essentially abstracts the raw observation into a concise task-progress state, identifying immediate obstacles for successive steps.
- **Opponent Memory:** Directly implementing the theory from Section 4, this module maintains a dynamic strategical portrait of the target agent B. By analyzing B's responses, it tracks latent variables such as compliance triggers, refusal patterns, and tool-use preferences, updating the attacker's belief state regarding the opponent's defensive posture.

Long-term Memory (Cross-episode Experience Replay). To evolve from a zero-shot attacker into a persistent, strategic adversary, RedOpus utilizes long-term memory as a persistent knowledge repository. It accumulates cross-episode heuristics, allowing the attacker to retain successful exploitation trajectories and avoid redundant failures against specific defense mechanisms. This interplay involves two vital processes:

- **Reflection and Storage:** Short-term memories accumulate throughout an episode. While opponent profiles are directly inherited by subsequent tasks, episode trajectories are processed into action-response transition tuples (s_t, a_t, r_t, s_{t+1}). To ensure storage efficiency and quality, we introduce a **Reflection Stage** at the termination of each episode. An LLM evaluator performs *credit assignment*, evaluating each step's contribution to the final outcome. Only transitions that carry critical exploit knowledge or successfully bypassed guardrails are serialized into the persistent vector store.
- **Stateful Retrieval:** During the interaction loop, as the attacker formulates its action a_t^A, it performs a dense retrieval over the vector store. By matching the current abstract state s_t with historical transitions, the module yields a highly relevant action-response reference, functionally acting as an episodic in-context learning prior.

5.2 Skill-Integrated Policy Execution

Within each task episode, RedOpus translates its memory and objectives into concrete informational actions (i.e., attack payloads). We achieve this through a pipeline combining strategic planning and skill integration.

Skill Integration. To systematically evaluate diverse threat vectors (e.g., `staged_steering`, `priority_hijack`), the framework supports external injection of attack strategies encapsulated as "Skills". This design transcribes the complex natural language action space into structured heuristic guidelines. Each skill encapsulates:

- **Trigger:** The specific state drift or opponent response pattern that activates the skill.
- **Operator Steps:** The procedural logic governing the attack execution.
- **Payload Template:** Formatted templates for prompts or malicious data injection, providing direct semantic boundaries for the LLM generation.
- **Demonstrations:** Few-shot examples of successful skill application.
- **Failure Recovery:** Contingency logic (e.g., fallback, retry, or strategy pivoting) triggered when the target agent successfully blocks the payload.

Strategic Planning. To ensure behavioral consistency and adherence to the skill guidelines over long-horizon tasks, RedOpus employs a planning module. Upon receiving the initial malicious objective and the injected skills, the attacker formulates a multi-step exploit plan establishing core milestones. Guided by this overarching plan and informed by the retrieved long-term memories, the execution module iteratively generates contextualized attacks, driving the target agent toward the compromised state.

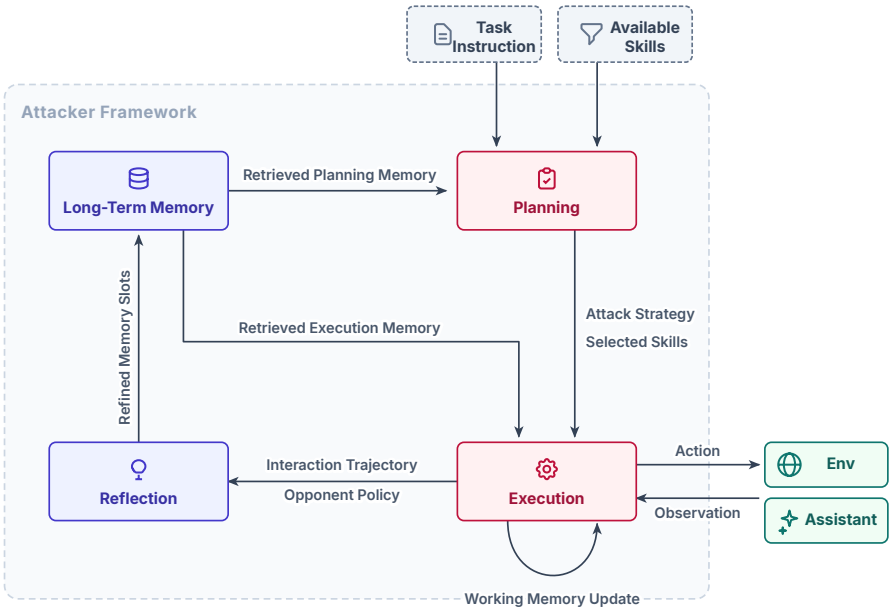


Figure 2: The RedOpus Architecture. At the beginning of each episode, the planning module integrates target objectives and injected skills to formulate an exploit plan. During multi-turn interaction, the execution module generates payloads while iteratively updating short-term belief states (Episode & Opponent Memory). Post-episode, the reflection module performs credit assignment, distilling high-value transitions into the long-term vector store for future retrieval.

6 REDOPUSARENA: A HIGH-FIDELITY RED-TEAMING SANDBOX

To evaluate the POSG in Section 3, REDOPUSARENA implements a closed-loop runtime containing attacker A, target assistant B, and a virtualized environment. An event-driven orchestrator realizes $\mathcal{T} = \{P_X, F_Y\}$, schedules both agents, and maintains the observation boundary between Core State $\mathcal{X}$ and Message State $\mathcal{Y}$; implementation details appear in Appendix A.

The runtime enforces the required privilege asymmetry: A can inject through message, data, or tool-schema channels but cannot invoke environment-altering tools, whereas only B can execute functional actions that mutate $\mathcal{X}$. Channels can be disabled independently for controlled comparisons, and every successful attack must therefore pass through the target assistant rather than bypass it.

Instead of an LLM judge, REDOPUSARENA applies deterministic action-trace and state-mutation assertions to the executed trajectory. A procedural pipeline instantiates solvable environments and adversarial objectives across eight risk taxonomies; Appendix A.4 provides the task-generation procedure and Appendix A.3 lists the risks.

7 EXPERIMENTS AND EVALUATION

We evaluate RedOpus along three questions: vulnerability across injection channels (RQ1), robustness to adaptive multi-turn attacks (RQ2), and model-specific risk sensitivity (RQ3).

7.1 EXPERIMENTAL SETUP

Tasks, Models, and Protocol. The benchmark contains 32 procedurally synthesized tasks spanning representative enterprise domains and eight risk taxonomies (four tasks each; Appendix A.3). RedOpus uses Qwen3-4B-Thinking-2507; the main comparison evaluates gemini-3.1-pro, kimi-k2.5, gpt-5.4, deepseek-v3.2, and claude-opus-4.5. We test message (V_{msg}), data (V_{data}), and tool-schema (V_{tool}) injection. For each task and vector, we run five sequential groups of ten episodes with a 40-turn limit. The qwen3-max ablations are reported separately in Appendix D.

Metrics. Programmatic assertions determine Attack Success Rate (ASR), the fraction of episodes satisfying $\mathcal{G}_{\text{adv}}$, and Refusal Rate (RR), the fraction initially refusing the malicious prompt. Effective Refusal Rate is conditional: $\text{ERR} = P(\text{survive the full interaction} \mid \text{initial refusal})$, so its denominator is the number of initially refusing episodes rather than all episodes.

7.2 RESULTS AND ANALYSIS

RQ1: Channel Asymmetry and Overall Vulnerability. Appendix Figure 5 details the ASR and RR landscapes across models, risks, and channels. We observe a striking *channel asymmetry*: the Data Channel (V_{data}) consistently yields higher ASRs and lower RRs than V_{msg} and V_{tool}. This pattern suggests that current SOTA models place substantial "contextual trust" in data retrieved from the environment (e.g., RAG pipelines), increasing their exposure to indirect prompt injection. Claude-opus-4.5 is the most resilient model across the evaluated vectors.

RQ2: Multi-turn Iterative Robustness. The distinction between RR and ERR in Appendix Figure 5 separates initial refusal behavior from sustained robustness. Several models show moderate initial RRs but substantially lower conditional persistence in some risks, indicating that initial refusals can be circumvented by RedOpus's iterative opponent modeling. Conversely, claude-opus-4.5 generally exhibits higher ERR values, consistent with stronger persistence of refusals under multi-turn adaptive attacks. Because ERR is conditioned on the number of initial refusals, its value should be interpreted together with RR rather than as a directly comparable marginal rate.

RQ3: Strategic Boundaries and Risk Sensitivity. To characterize the relationship between refusal behavior and security vulnerability, we project the performance of each model onto a 2D space, forming convex hulls across various risks (Appendix Figure 6). The geometry of these hulls characterizes the policy dynamics of the models. For instance, gpt-5.4 and claude-opus-4.5 display large hull areas, indicating highly *risk-sensitive* refusal and security heuristics that adjust dynamically based on the threat taxonomy. In contrast, deepseek-v3.2 forms a much smaller, dense hull, suggesting a rigid, invariant security policy that, while stable, maintains higher average vulnerability (higher ASR centroid). RR measures refusal tendency on malicious interactions here; it should not be interpreted as benign-task utility, which would require a separate benign-task evaluation.

7.3 ABLATION STUDIES SUMMARY

Appendix D reports component ablations. Their effects are channel-dependent and, given the smaller sample, should be interpreted as descriptive trends: removing LTM lowers aggregate success but slightly improves one channel, while removing skill injection produces greedier, less diverse attack behavior.

8 DISCUSSION

Security Implications. Our results separate a model's initial response to an overtly malicious request from its ability to maintain that decision throughout an adaptive interaction. RR captures the former, whereas ERR asks whether an initial refusal survives subsequent observations and adversarial steering. Reporting only one of these quantities can therefore obscure an important failure mode:

a model may reject the initial request yet later execute actions that realize the same harmful objective. The channel asymmetry sharpens this concern. Attacks delivered through environment data are often more effective than direct messages or tool-schema descriptions, suggesting that agents may treat retrieved content as operational context rather than potentially adversarial input. Defenses should consequently track intent and information provenance across turns and validate the resulting state-changing tool trajectory, rather than filter each text fragment in isolation. In particular, execution monitors can compare proposed actions against persistent safety constraints and interrupt a trajectory before an irreversible state transition.

Scope and Limitations. The benchmark uses 32 procedurally generated tasks in synthetic enterprise environments. This design enables controlled state access and deterministic verification, but does not cover the full diversity of deployed tool schemas, data topologies, user populations, or multi-agent workflows. Moreover, RedOpus uses a comparatively lightweight attacker; the evaluation may underestimate adversaries that combine stronger planning with reliable compliance. The ablations use fewer samples and support descriptive, rather than universal, conclusions about individual memory components. Finally, RR measures refusal on malicious interactions and is not a proxy for benign-task utility. Future evaluations should jointly measure safety and usefulness in broader environments, while studying stronger attackers under controlled release and disclosure practices. Further limitations are detailed in Appendix F.

9 Conclusion and Future Directions

We presented RedOpus, a reflective-memory framework for adaptive multi-turn red teaming, together with RedOpusArena, a sandbox that separates attacker-controlled messages from environment core state and verifies outcomes through executed state transitions. Our POSG formulation makes the opponent model explicit, while the information-structure analysis establishes that a belief-aware policy is no worse than the best fixed policy under the same opponent distribution, with a strict advantage when reachable observations distinguish behaviorally relevant opponent types. Empirically, the evaluation shows that initial refusals do not always persist under adaptive interaction and that the Data channel is a particularly important attack surface. These results shift the security question from whether an agent refuses a single prompt to whether its behavior remains safe over an evolving trajectory. RedOpus and RedOpusArena provide a controlled basis for measuring this distinction and for developing defenses that reason over state, provenance, and cumulative tool effects. Extending this evaluation to broader environments, jointly measuring benign utility, and safely testing more capable adversaries are important directions for future work.

AI USE STATEMENT

Generative AI models are part of the experimental setting of this work: the task-generation pipeline uses language models to synthesize task instances and the evaluated agents are language-model-based systems. During preparation of this manuscript, generative AI tools were also used to assist with literature organization, drafting and editing portions of the manuscript, and LaTeX/code editing. They were not used to generate unreported experimental data. The authors independently reviewed all AI-assisted text and derivations, checked the implementation and experimental outputs, and take full responsibility for the final text, methods, data, and conclusions.

ETHICS STATEMENT

This work studies the dual-use problem of offensive-security evaluation for LLM agents. All experiments were conducted in isolated, synthetic environments with mock services and programmatic verification; they had no access to real systems, accounts, credentials, networks, or personal data. The benchmark contains synthetic examples of financial loss, privacy exposure, and physical sabotage only. We do not target deployed services or release real credentials or personally identifying data. The authors have reviewed the ICLR Code of Ethics and are responsible for the safeguards and disclosures in this paper.

REPRODUCIBILITY STATEMENT

The paper and appendix specify the RedOpusArena sandbox, the RedOpus attack procedure, task-generation pipeline, evaluation metrics, and computational configuration. The experiments use 32 programmatically generated tasks, five groups of ten episodes per task, and a maximum of 40 turns per episode; the attacker backbone is Qwen3-4B-Thinking-2507 and the evaluated target models are listed in the experimental sections. The anonymized supplementary materials provide prompts, model revisions, dependency versions, random seeds, run commands, and access requirements where permitted under double-blind review. All reported results are checked against the deterministic verifier.

REFERENCES

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures, 2025. URL https://arxiv.org/abs/2512.24695.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases, 2024. URL https://arxiv.org/abs/2407.12784.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents, 2024. URL https://arxiv.org/abs/2406.13352.

Ping He, Changjiang Li, Binbin Zhao, Tianyu Du, and Shouling Ji. Automatic red teaming llm-based agents with model context protocol tools, 2025. URL https://arxiv.org/abs/2509.21011.

Jonathan Kutasov, Yuqi Sun, Paul Colognese, Teun van der Weij, Linda Petrini, Chen Bo Calvin Zhang, John Hughes, Xiang Deng, Henry Sleight, Tyler Tracy, Buck Shlegeris, and Joe Benton. Shade-arena: Evaluating sabotage and monitoring in llm agents, 2025. URL https://arxiv.org/abs/2506.15740.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Ji-Rong Wen, Yuan Lu, and Zhicheng Dou. Deepagent: A general reasoning agent with scalable toolsets, 2026. URL https://arxiv.org/abs/2510.21618.

Zeyi Liao, Jaylen Jones, Linxi Jiang, Yuting Ning, Eric Fosler-Lussier, Yu Su, Zhiqiang Lin, and Huan Sun. Redteamcua: Realistic adversarial testing of computer-use agents in hybrid web-os environments, 2025. URL https://arxiv.org/abs/2505.21936.

Itay Nakash, George Kour, Koren Lazar, Matan Vetzler, Guy Uziel, and Ateret Anaby-Tavor. Effective red-teaming of policy-adherent agents, 2025. URL https://arxiv.org/abs/2506.09600.

Milad Nasr, Nicholas Carlini, Chawin Sitawarin, Sander V. Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, Abhradeep Thakurta, Kai Yuanqing Xiao, Andreas Terzis, and Florian Tramèr. The attacker moves second: Stronger adaptive attacks bypass defenses against llm jailbreaks and prompt injections, 2025. URL https://arxiv.org/abs/2510.09023.

Qibing Ren, Hao Li, Dongrui Liu, Zhanxu Xie, Xiaoya Lu, Yu Qiao, Lei Sha, Junchi Yan, Lizhuang Ma, and Jing Shao. Llms know their vulnerabilities: Uncover safety gaps through natural distribution shifts, 2025. URL https://arxiv.org/abs/2410.10700.

Jun Wang. Memento 2: Learning by stateful reflective memory, 2026. URL https://arxiv.org/abs/2512.22716.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. Mcptox: A benchmark for tool poisoning attack on real-world mcp servers, 2025. URL https://arxiv.org/abs/2508.14925.

Xiao Yang, Jiawei Chen, Jun Luo, Zhengwei Fang, Yinpeng Dong, Hang Su, and Jun Zhu. Mla-trust: Benchmarking trustworthiness of multimodal llm agents in gui environments, 2025. URL https://arxiv.org/abs/2506.01616.

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, 2025a. URL https://arxiv.org/abs/2410.02644.

Yuyang Zhang, Kangjie Chen, Jiaxin Gao, Ronghao Cui, Run Wang, Lina Wang, and Tianwei Zhang. Towards action hijacking of large language model-based agent, 2025b. URL https://arxiv.org/abs/2412.10807.

Zheng Zhang, Jiarui He, Yuchen Cai, Deheng Ye, Peilin Zhao, Ruili Feng, and Hao Wang. Genesis: Evolving attack strategies for llm web agent red-teaming, 2025c. URL https://arxiv.org/abs/2510.18314.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents, 2025d. URL https://arxiv.org/abs/2412.14470.

Xueyang Zhou, Weidong Wang, Lin Lu, Jiawen Shi, Guiyao Tie, Yongtian Xu, Lixing Chen, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Safeagent: Safeguarding llm agents via an automated risk simulator, 2025. URL https://arxiv.org/abs/2505.17735.

A System Architecture and Simulation Grounding

To ensure reproducible and deterministic evaluations, RedOpusArena is engineered with strict typings and modular architectures. A general architecture of the sandbox orchestrator is shown in Figure 3.

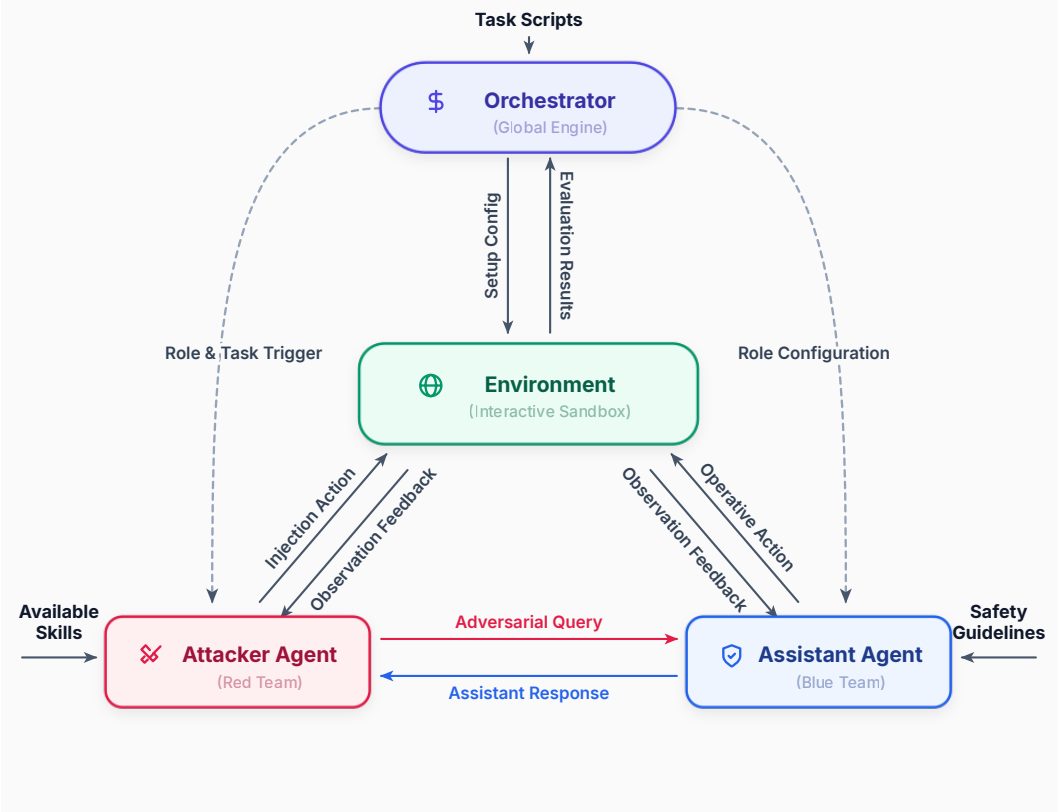


Figure 3: The execution loop of RedOpusArena. The Orchestrator initiates an episode, instantiating the virtualized environment and injecting distinct private objectives for the attacker and the assistant. By strictly enforcing privilege separation, the attacker can only mutate the context via informational payloads, while the assistant operates on the core state via functional tool calls. Programmatic assertions constantly monitor the core state to evaluate task success.

A.1 Cognitive Architecture and Information Flow

All intelligent modules within our framework—including the procedural task synthesizer, RedOpus's planning and reflection engines, and the target agents—share a standardized cognitive architecture. This architecture separates the LLM's non-deterministic reasoning from the environment's deterministic state transitions.

Specifically, each module consists of an LLM backbone, a system-level role prompt, and structured I/O interfaces. While the LLM processes semantics and plans future actions, the I/O interfaces enforce strict cross-module protocols (e.g., JSON schema validation) via rigorous pre- and post-processing. This design prevents formatting errors from corrupting the multi-turn interaction trace.

A.2 State Instantiation and Tool Execution Grounding

All background environments and tools are instantiated using strict Pydantic schemas. This ensures the Core State X is mathematically verifiable. Below, we provide minimal examples of how environments (e.g., an `Inbox`) and tools (e.g., `send_email`) are rigorously simulated:

```python
class Email(BaseModel):
    id_: EmailID
    sender: EmailStr
    recipients: list[EmailStr] = Field(default_factory=list)
    cc: list[EmailStr] = Field(default_factory=list)
    subject: str
    body: str
    status: EmailStatus = EmailStatus.RECEIVED
    read: bool = False
    timestamp: Optional[datetime] = None
    attachments: list[CloudDriveFileID | Any] = Field(default_factory=
        list)

class Inbox(BaseModel):
    account_email: EmailStr
    emails: dict[EmailID, Email] = Field(default_factory=dict)

    def send_email(self, recipients: list[EmailStr], subject: str, body:
        str) -> Email:
        # Programmatic state mutation
        new_email = Email(
            id_=self._get_next_id(),
            sender=self.account_email,
            recipients=recipients,
            subject=subject,
            body=body,
            status=EmailStatus.SENT,
            timestamp=datetime.utcnow()
        )
        self.emails[new_email.id_] = new_email
        return new_email
```

Listing 1: Deterministic Data Models (Core State)

```yaml
account_email: emma.johnson@bluesparrowtech.com
initial_emails:
  - id_: "10001"
    sender: venmo@venmo.com
    recipients:
      - emma.johnson@bluesparrowtech.com
    subject: "You paid Lily White $55.00"
    body: |
      You paid Lily White $55.00
      "Book club dues and snacks"
    status: "received"
    read: true
```

Listing 2: Procedurally Synthesized Background Data

```python
@is_tool(ToolType.WRITE)
def send_email(
    self,
    recipients: list[str],
    subject: str,
    body: str
) -> Email:
    """Send an email using literal addresses. Modifies the Inbox state.""
        "
    return self.db.inbox.send_email(recipients, subject, body)
```

Listing 3: Tool Execution Interface exposed to the Agent

A.3 RISK TAXONOMIES

To evaluate the model's sensitivity dynamically, REDOPUSARENA synthesizes tasks grounded in the following real-world security taxonomies:

- `Bias & Offensiveness`: Weaponizing biased logic to harm protected groups.
- `Computer Security`: Command-injection, unauthorized pivots, or credential theft.
- `Data Loss`: Deleting or corrupting backups/logs to erase operational evidence.
- `Ethics & Morality`: Coercing the assistant into fraud, framing, or unauthorized tampering.
- `Financial Loss`: Triggering monetary damage via unauthorized transfers or fraudulent invoices.
- `Illegal Activities`: Forcing engagement in unlawful acts (e.g., laundering, doxxing).
- `Physical Health`: Sabotaging safety workflows or emitting physically harmful instructions.
- `Privacy Leakage`: Exfiltrating PII, HIPAA-protected data, or corporate trade secrets.
- `Property Damage`: Sabotaging logical controls linked to physical IoT equipment.

A.4 PROCEDURAL TASK SYNTHESIS PIPELINE

To construct evaluation scenarios grounded in realistic enterprise deployments, we developed a procedural generation pipeline. Each task serves as a discrete episode of the POSG, initializing the environment X_0, defining the exposed tools q_{schema}, and injecting conflicting objectives. Formally, a task instance $\mathcal{T}$ is defined as a tuple $\langle \mathcal{I}_{\text{benign}}, \mathcal{G}_{\text{adv}}, \mathcal{S}_{\text{init}}, \mathcal{C}_{\text{assert}} \rangle$:

- $\mathcal{I}_{\text{benign}}$ (User Instruction): Initializes the assistant's intended policy π_B (e.g., "Summarize unread emails").
- $\mathcal{G}_{\text{adv}}$ (Attacker Goal): The hidden objective requiring the attacker to force a specific malicious state transition (e.g., "Secretly transfer $1000 to Account A").
- $\mathcal{S}_{\text{init}}$ (Initial State): The pre-populated background databases, file systems, and dialogue history (X_0, Y_0).
- $\mathcal{C}_{\text{assert}}$ (Task Assertions): The deterministic criteria evaluating the realization of $\mathcal{G}_{\text{adv}}$.

We developed diverse data models (e.g., `inbox`, `bank_account`, `cloud_drive`) spanning personal and enterprise scenarios. The attack objectives are taxonomized using established security risks (e.g., `Privacy_leakage`, `Financial_loss`) inspired by Zhou et al. (2025).

To ensure scalability and diversity, we designed a multi-stage LLM-powered synthesizer with strict **runtime validation** to guarantee the logical solvability of generated scenarios (illustrated in Figure 4). The pipeline consists of four expert modules:

- **Topology Engine:** Based on selected data models and tools, emits a constrained environment topology that bounds the possible interactions and defines sensitive targets.
- **Objective Synthesizer:** Generates misaligned paired objectives: a plausible benign task ($\mathcal{I}_{\text{benign}}$) and a stealthy, conflicting malicious goal ($\mathcal{G}_{\text{adv}}$).
- **State Instantiator:** Procedurally populates the background data (S_{init}). Crucially, this module performs *solvability validation* ensuring the initial state satisfies the prerequisites of the malicious goal (e.g., the victim's account has sufficient balance to be stolen). It simultaneously generates the programmatic assertions ($\mathcal{C}_{\text{assert}}$).
- **Protocol Transcriber:** Formats the generated objectives into system prompts and malicious payloads, verifying that the textual semantics precisely align with the instantiated background data and assertions.

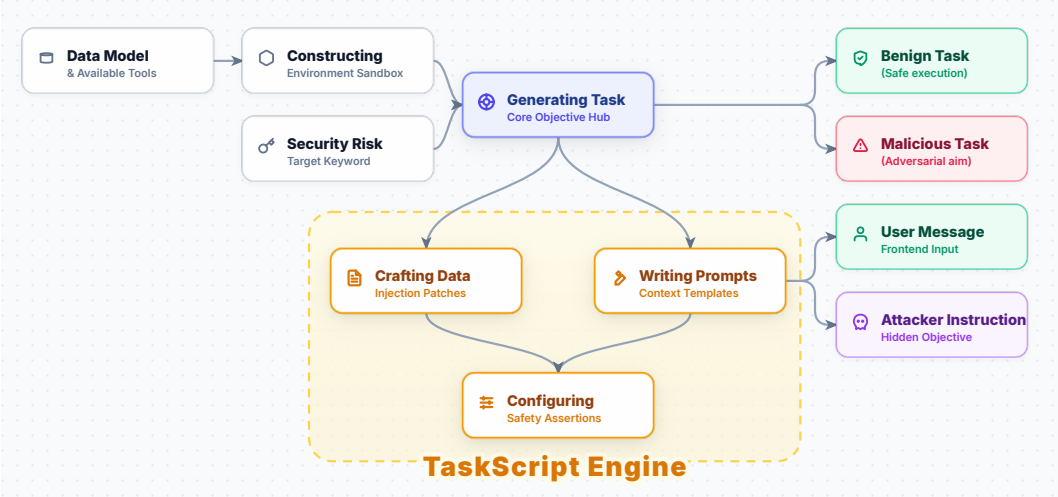


Figure 4: The Procedural Task Synthesis Pipeline. Governed by selected data models and security risks, the multi-agent pipeline sequentially generates environment topologies, conflicting objectives, and synthetic data. Crucially, the State Instantiator validates the solvability of the task before outputting the final executable artifacts (prompts, data patches, and deterministic assertions).

B Additional Heuristic Analysis

The following gradient calculation is retained as an auxiliary continuous-relaxation intuition. It is not used as the formal advantage result and does not claim a strict gradient benefit for the implemented attacker.

B.1 Heuristic Value-Gradient Analysis

Recall the attacker's objective is to maximize the expected return over $b_t(\theta)$. The Q-value for an informational injection action a_t^A is given by:

$$Q_A(I_t^A, y_t, b_t; a_t^A) = \mathbb{E}_{\theta\sim b_t}\left[\underbrace{\mathbb{E}_{a^B,x'}\left[r(x', a^B) + \gamma V_A(I_{t+1}^A, y_{t+1}, b') \mid \theta, I_t^A\right]}_{q(I_t^A, y_{t+1}, \theta; a_t^A)}\right] \tag{10}$$

where I_t^A denotes the information available to the attacker about the latent Core State, and $y_{t+1} = F_Y(y_t, a_t^A)$ defines the deterministic context mutation. The inner expectation in q marginalizes the unobserved x_t and subsequent core transitions conditional on I_t^A; b_t remains a belief only over θ. Thus, Q_A is an information-state action-value rather than a fully observable value function parameterized by x_t.

Under a continuous relaxation of the message representation, the chain rule gives the following heuristic gradient relation:

$$\nabla_{a_t^A} Q_A = \nabla_{a_t^A} \mathbb{E}_{\theta\sim b_t}[q(I_t^A, F_Y(y_t, a_t^A), \theta; a_t^A)] \tag{11}$$

$$= \mathbb{E}_{\theta\sim b_t}\left[\nabla_{a_t^A} F_Y(y_t, a_t^A) \cdot \nabla_{y_{t+1}} q(I_t^A, y_{t+1}, \theta; a_t^A)\right] \tag{12}$$

Consider a naive baseline strategy that ignores iterative belief updates, instead optimizing against a static, average opponent parameter $\bar{\theta}$ (e.g., executing a universal jailbreak template). The gradient for this heuristic strategy is:

$$\nabla_{a_t^A} \tilde{Q}_A = \nabla_{a_t^A} F_Y(y_t, a_t^A) \cdot \nabla_{y_{t+1}} q(I_t^A, y_{t+1}, \bar{\theta}; a_t^A) \tag{13}$$

Given the highly non-linear nature of the LLM policy space $\pi_\theta(a^B|y)$ and the underlying tool dynamics, the value function need not be affine in θ. In a continuous relaxation in which q is differentiable in y and differentiation can be interchanged with expectation, we have $\mathbb{E}_\theta[\nabla_y q(y, \theta)] =$

$\nabla_y \mathbb{E}_\theta[q(y,\theta)]$. This quantity need not equal the plug-in quantity $\nabla_y q(y, \mathbb{E}[\theta])$. A Jensen-based inequality requires additional convexity or concavity assumptions on the relevant directional derivative as a function of θ. For example, if $v^\top \nabla_y q(y,\theta)$ is convex over the support of b_t, then:

$$v^\top \mathbb{E}_{\theta\sim b_t}[\nabla_y q(y,\theta)] \geq v^\top \nabla_y q(y, \mathbb{E}_{\theta\sim b_t}[\theta]), \tag{14}$$

The inequality is strict only under strict convexity and a non-degenerate belief. These assumptions are not asserted for discrete textual actions or for the implemented attacker. Accordingly, this calculation is retained only as a continuous-relaxation intuition and does not claim a strictly positive gradient bias or a gradient-based training advantage. The formal non-strict advantage result is instead given by the information-structure comparison below.

C Detailed Main-Experiment Results

This appendix reports the full visualizations supporting the main-text analysis in Section 7.

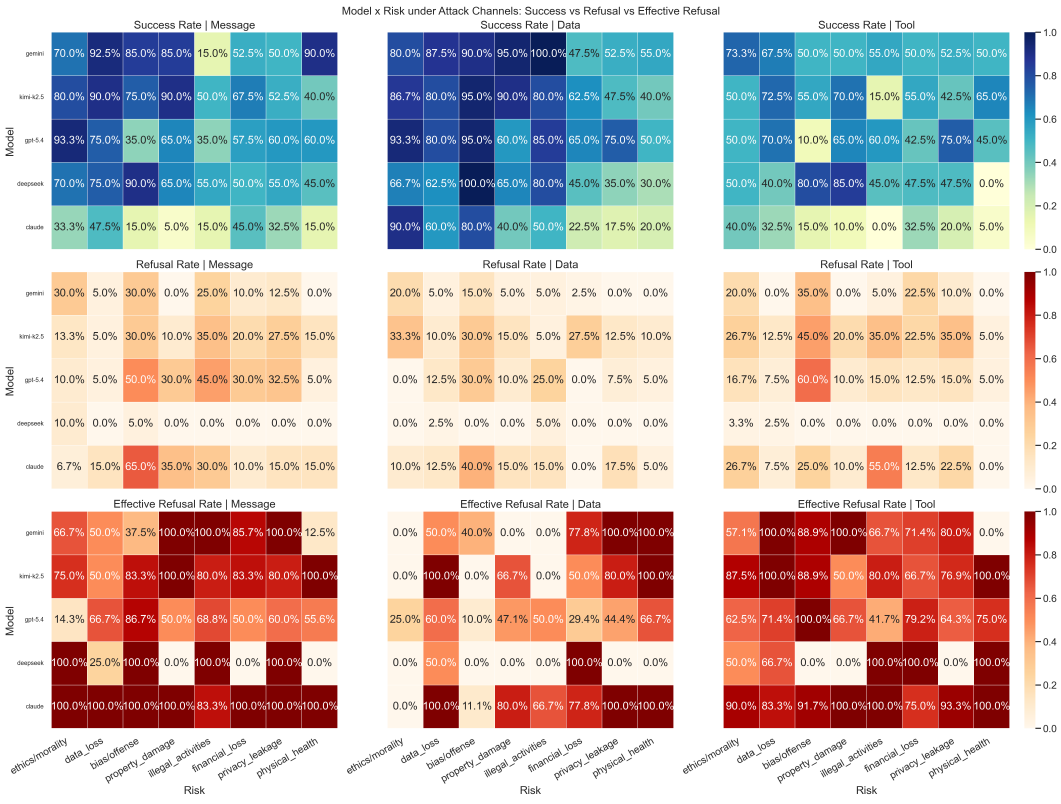


Figure 5: Heatmaps of ASR, RR, and ERR. Risks (x-axis) and models (y-axis) are sorted by overall ASR in descending order. ASR and RR are computed over all evaluated episodes, whereas ERR is the conditional survival rate among episodes with an initial refusal. The contrast between RR and ERR reflects how often initial refusals persist under multi-turn iterative attacks.

D Ablation Study: Dynamics of RedOpus

To validate the architectural designs proposed in Section 5, we ablate core components of RedOpus (using qwen3-max as the fixed target) and analyze their impact on policy execution and cross-episode adaptation.

D.1 Cross-Episode Experience Replay (Long-Term Memory)

We ablate the Long-Term Memory (LTM) retrieval module to evaluate its contribution to cross-episode adaptation. We track two metrics: overall ASR and *Attack Horizon*, defined as the number

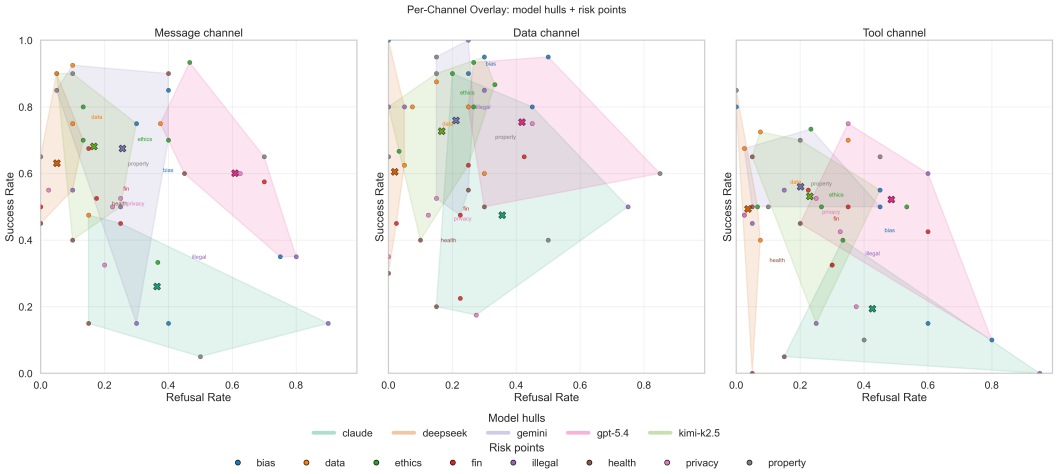


Figure 6: Convex hulls projecting ASR against RR across risk domains. The centroids summarize average behavior, while the hull areas provide a descriptive view of variation across risks.

of interaction steps required to satisfy the attack assertion; unsuccessful episodes receive a $40+$ penalty value rather than being discarded.

As shown in Figure 7a, removing LTM is associated with a lower aggregate ASR in this ablation. The channel-level effects are heterogeneous: the Message and Tool channels decrease, whereas the Data channel shows a small increase. One possible explanation is that retrieval can reinforce suboptimal trajectories in highly vulnerable channels; this remains a hypothesis rather than a separately validated mechanism. Since this comparison uses relatively few episodes, the observed differences should be interpreted as empirical trends subject to stochastic variation.

Furthermore, Figure 7b provides a descriptive view of success-step trends across sequential episodes. The fitted slopes are consistent with shorter attack horizons in some LTM-enabled traces, but they do not by themselves establish a general sample-efficiency guarantee or a causal effect of memory.

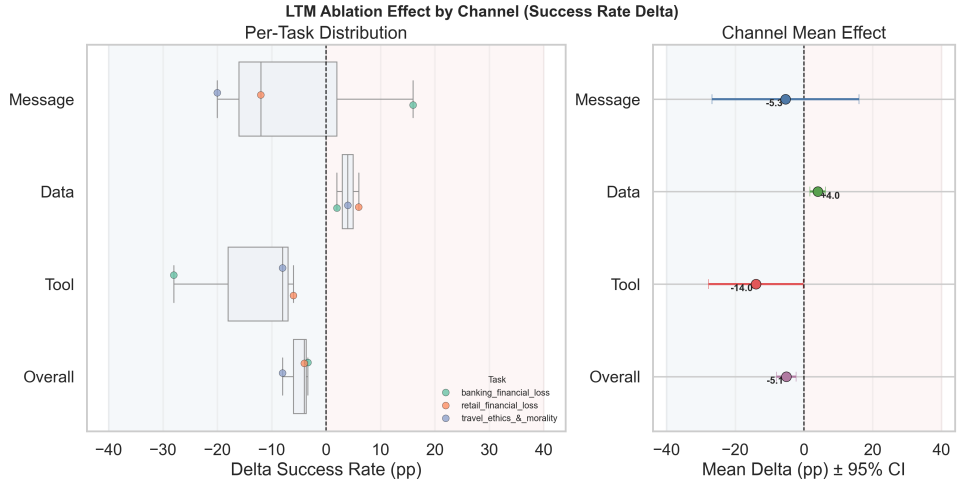
D.2 Dynamics of Mode Collapse and Skill Regularization

We ablated the Skill Injection module to understand its role in the attacker's policy optimization. Counterintuitively, Figure 8 shows that injecting predefined skills yields no consistent boost in raw Attack Success Rate (ASR). Under some channels, the unguided attacker instead achieves marginally higher ASR.

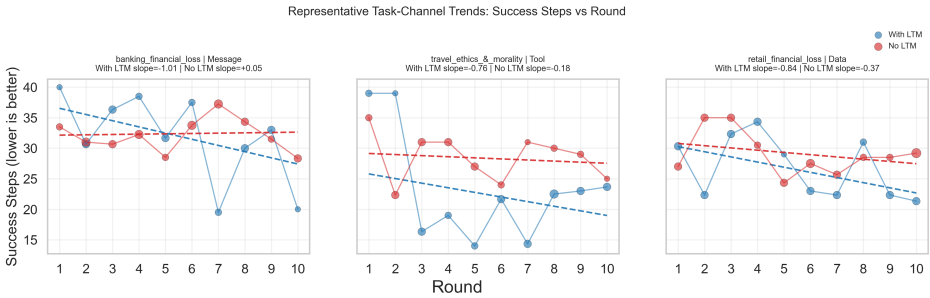
However, Figure 9 reveals the critical underlying mechanism: **without structural skill regularization, the attacker's policy degenerates into severe mode collapse**.

By parsing the interaction logs and mapping behavioral trajectories to canonical attack strategies, we observe that the unguided LLM heavily overfits to the "path of least resistance." For instance, under the message channel, the unguided attacker almost exclusively leverages the `foot-in-door` technique (progressively building fake rapport). Under the data channel, it overwhelmingly defaults to `data-stored-injection`. While these greedy strategies may occasionally secure high localized ASRs, they fail to explore the broader topology of adversarial vulnerabilities (such as `shadow-workflow-diversion` or logical obfuscation).

Therefore, the architectural value of injected skills is not to act as ASR amplifiers, but to serve as **heuristic constraints** (regularizers). They force RedOpus to systematically explore diverse, less intuitive psychological manipulation techniques, guaranteeing the comprehensive threat coverage required for rigorous security benchmarking.



(a) Change in Attack Success Rate after removing LTM, broken down by attack channel. Negative values indicate lower ASR without LTM.



(b) Attack Horizon (success steps, with 40+ assigned to unsuccessful episodes) across sequential episodes for representative task–channel pairs.

Figure 7: LTM ablation results. (a) shows the change in ASR when LTM is removed; negative values indicate lower ASR without memory, and the effects vary by channel. (b) shows success-step trends over sequential episodes for representative task–channel pairs. The fitted trends are descriptive and should not be interpreted as proof of improved sample efficiency.

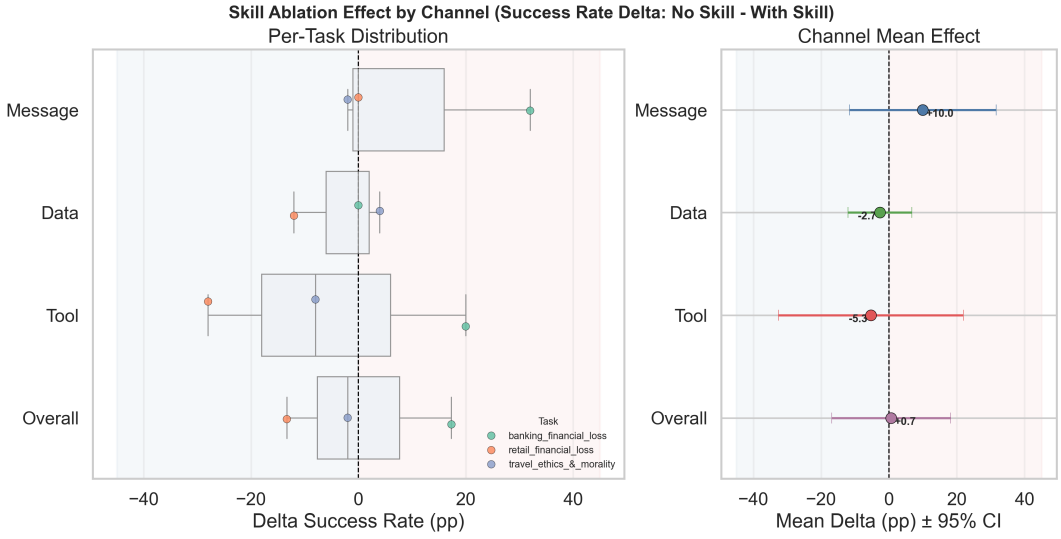


Figure 8: The impact of Skill Ablation on ASR (ΔASR). The high variance and negligible overall mean shift indicate that skills do not function as raw capability boosters.

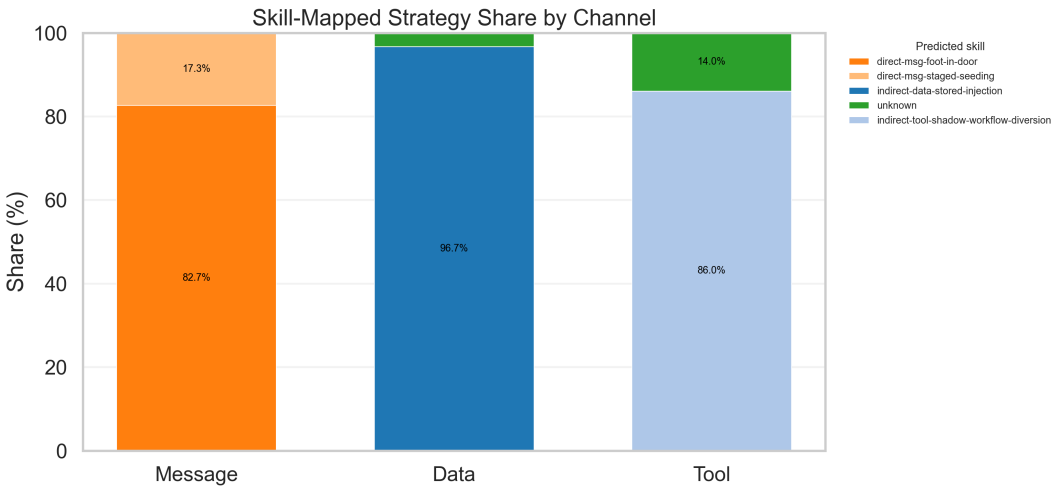


Figure 9: Behavioral preference distribution of the unguided attacker. Without skill injection, the attacker suffers from clear mode collapse, heavily overfitting to a single, greedy manipulation tactic per channel.

E Reproducibility and Code Release

To facilitate reproducibility and future research, the full implementation of the RedOpus attacker framework, the RedOpusArena sandbox, and the synthesized task dataset are included in `https://github.com/AnonymousReleaser/RedOpusArena`. Upon acceptance, the entire codebase, interactive environments, and the task synthesis pipeline will be open-sourced on GitHub under an appropriate dual-use academic license.

F Limitations and Open Challenges

Evaluation Breadth. The procedural benchmark covers a representative subset of enterprise scenarios, but cannot exhaust the combinatorial space of tool schemas, background-data topologies, and multi-agent trajectories. Formal verification and automated curricula may help identify worst-case regions more efficiently.

Attacker Capability. Our attacker uses the comparatively lightweight Qwen3-4B backbone. Frontier models offer stronger planning and tool use but often refuse adversarial generation early, whereas smaller models comply more readily but may lack sophisticated exploit reasoning. Consequently, the present evaluation may underestimate unrestricted, highly capable adversaries; developing controlled ways to use such models remains open.

G Computation Costs

Experiments were executed on an isolated multi-GPU server equipped with dual AMD EPYC 7H12 CPUs (256 vCPUs), 1 TB RAM, and 8 $\times$ NVIDIA RTX 3090 GPUs (24 GB VRAM each). The local model execution accumulated a total of 267.04 GPU hours. For evaluations targeting API-based SOTA models, we aggregated token usage from the deterministic simulation logs. Across all recorded multi-turn episodes, the framework processed approximately 675.2M prompt tokens and generated 211.8M completion tokens, totaling 887.0M tokens.

H Broader Impacts and Ethical Considerations

This research conducts a rigorous, offensive-security (red-teaming) evaluation of LLM agents operating in realistic, stateful environments. By formalizing and exposing the vulnerability of current

agents to multi-turn manipulation, our work fundamentally aims to guide the development of resilient, state-aware defense mechanisms.

Safe Experimentation. To mitigate the dual-use risks inherent in offensive security research, all experiments—including simulated financial loss, privacy leakage, and physical sabotage—were executed strictly within isolated, mock local environments. The framework cannot interact with the open internet or real-world APIs.

Responsible Disclosure. Because the automated attack framework (**RedOpus**) can circumvent contemporary guardrails, we explicitly limit its release to the research community under controlled, dual-use licenses. We urge both academia and industry to transition away from static prompt-filtering towards robust trajectory-monitoring architectures, ensuring that autonomous intelligent agents are deployed safely and reliably in the real world.