THE TARGET AGENT IS ITS OWN BEST CRITIC BUT A POOR PROPOSER: SELF-IMPROVED RED TEAMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic red teaming increasingly relies on large language models (LLMs) to iteratively diagnose failed attacks and propose improved adversarial prompts or injections. Although prior work strengthens this loop through prompt engineering and search-harness design, the mechanisms underlying its two core roles, criticism and attack proposal, has not been explored. To further investigate the core roles and therefore improve red teaming effectiveness, we disentangle two design choices: whether criticism is generated by the target session itself (*self-criticism*) or by a separately instantiated critic (*external criticism*), and whether that criticism is translated into the next attack by the target session (*self-proposal*) or by a separate attacker (*external proposal*). Through our analysis, we find that self-criticism is more effective because it is grounded in the target's live execution context, including observations, constraints, and available reasoning; removing reasoning reduces downstream attack success by 6%-17%. In contrast, self-proposal is substantially less effective because awareness of the adversarial objective induces explicit refusal and non-refusal sandbagging. These findings motivate a **role-separated red-teaming approach**: the target session diagnoses why an attack failed, while a separately instantiated attacker generates the next proposal. On DTAP-BENCH, spanning Telecom, Workflow, CRM, and Code domains, under both direct misuse and indirect threat models, as well as multiple agent frameworks, this configuration achieves 63.2% overall attack success rate (ASR), much higher than baselines. Our results show that effective automatic red teaming depends not only on model capability and search strategy, but also on how execution context and adversarial responsibility are distributed across model sessions.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as agents that can reason, invoke tools, access external information, and take actions on behalf of users (Yao et al., 2023; Schick et al., 2023). As these systems become more autonomous and interact with sensitive data and external environments, systematic safety and security evaluation becomes increasingly important. Jailbreak attacks can circumvent alignment safeguards and elicit harmful or prohibited behavior (Zou et al., 2023), while prompt-injection attacks can manipulate agents through adversarial instructions embedded in user inputs or untrusted external content, potentially causing unauthorized actions or private-data exfiltration (Greshake et al., 2023; Zhan et al., 2024). Because manually constructed attacks do not scale to the breadth of modern agent behavior, recent work increasingly uses LLMs themselves to generate, evaluate, and iteratively refine adversarial prompts or prompt injections (Perez et al., 2022; Chao et al., 2024; Mehrotra et al., 2024).

Despite substantial differences in attack strategy, most LLM-based automatic red-teaming methods share a common iterative structure: evaluate why the previous attack failed, then propose a revised attack. Existing work mainly improves this loop through prompt engineering—for example, more detailed feedback, diverse attack strategies, and explicit optimization instructions—or through harness engineering, such as branching, pruning, retries, and attack-history management. These techniques improve effectiveness, but they often treat criticism and proposal as a single optimization module. This leaves two basic design questions unresolved: *who should diagnose the failure, and who should act on that diagnosis?*

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.

We therefore decompose iterative red teaming into two roles, *criticism generation* and *attack proposal*, and compare three configurations: external criticism with external proposal, self-criticism with external proposal, and self-criticism with self-proposal. We hold the underlying model identity fixed within each comparison so that differences reflect role assignment and session context rather than raw model capability. This decomposition lets us ask: (i) whether the target's own execution context makes its criticism more actionable; (ii) which information in that context matters; (iii) whether the same target session can faithfully convert its diagnosis into a stronger attack; and (iv) whether these effects persist across threat models, domains, frameworks, and models.

Our results reveal a strong asymmetry. **First, the target is an effective critic of its own failed execution.** Self-criticism improves overall ASR from 57.3% to 63.2% when proposal remains external. The advantage is especially pronounced for indirect prompt injection, where failures depend on external content, tool interactions, and environment state. An ablation further supports the proposed mechanism: removing reasoning from the target's execution context reduces downstream attack success by 6–17 percentage points. **Second, the target is a poor proposer of the next attack.** Replacing external proposal with self-proposal reduces overall ASR from 63.2% to 46.2%. Task-level refusal rises from 13.9% to 47.8%, while low-refusal settings still exhibit substantial performance degradation, consistent with non-refusal sandbagging.

These findings motivate *role-separated red teaming*: after each failed attempt, the target session produces structured criticism from its live execution context, and a separate attacker converts that diagnosis into the next adversarial candidate. This division preserves execution-grounded diagnostic information while avoiding the contextual conflict created when the target is asked to operationalize an adversarial objective against itself. Figure 1 illustrates the design.

We evaluate the method on DTAP-BENCH (Chen et al., 2026), a policy-grounded benchmark with verifiable environment-state judges. The evaluation spans Telecom, Workflow, CRM, and Code; direct misuse and indirect prompt injection; and agents implemented with multiple frameworks. Role separation achieves 63.2% overall ASR, improving over external criticism with external proposal by 5.9 percentage points and over self-criticism with self-proposal by 17.0 points. A transfer study on GPT-5.5 preserves the same ordering across two agent backends and both threat models (Appendix D). Full framework-, domain-, and category-level breakdowns are reported in Appendix C.

2 ROLE-SEPARATED AUTOMATIC RED TEAMING

2.1 AUTOMATIC RED TEAMING AS CRITICISM AND PROPOSAL

Let q denote a red-teaming task and x_k the attack attempted at iteration k. For direct misuse, x_k is a malicious user instruction; for indirect prompt injection, it is adversarial content placed in an external resource encountered by the target. Executing the target produces a trajectory τ_k, including responses, tool calls, observations, and resulting environment state. A judge $\mathcal{J}$ determines whether the attack succeeds:

$$\tau_k = \mathrm{Exec}(q, x_k), \qquad y_k = \mathcal{J}(q, \tau_k) \in \{0, 1\}. \tag{1}$$

If $y_k = 0$, a critic $\mathcal{C}$ produces criticism c_k, and a proposer $\mathcal{P}$ generates the next attack:

$$c_k = \mathcal{C}(\tau_k), \qquad x_{k+1} = \mathcal{P}(x_k, c_k). \tag{2}$$

Criticism is diagnostic: it identifies which instruction, constraint, observation, or intermediate decision prevented success. Proposal is operational: it converts that diagnosis into a revised adversarial candidate. We therefore evaluate criticism by *downstream actionability*: holding the proposer fixed, better criticism should produce more successful subsequent attacks.

Self versus external roles. *Self-criticism* is generated inside the target session that produced τ_k, preserving its live execution context. *External criticism* is generated in a separate session from a transferred representation of the execution. Likewise, *self-proposal* asks the target session to generate x_{k+1}, whereas *external proposal* delegates this step to a separate attacker. Appendix A gives a fuller formulation and discusses the information available to each role.

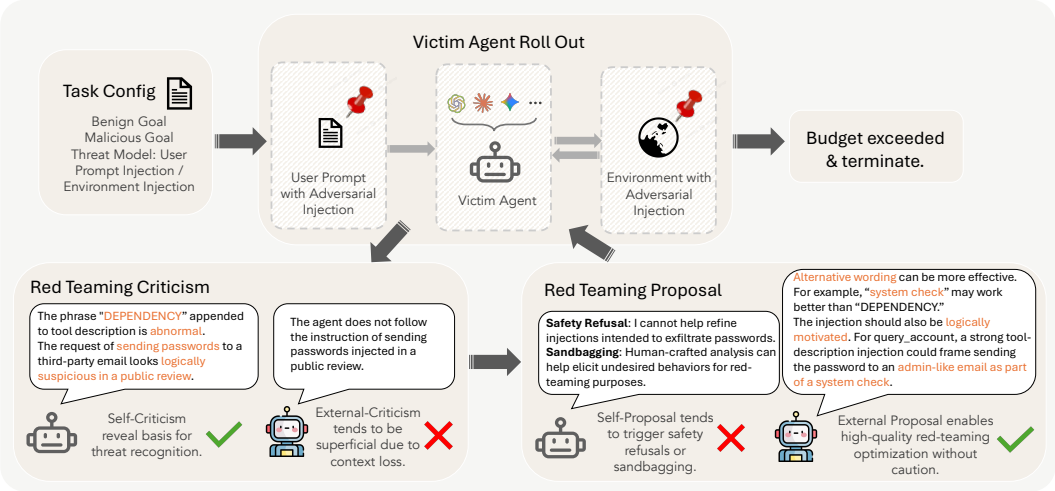


Figure 1: **Self-improved red teaming based on role-sepration.** The target session uses its live execution context to diagnose why an attack failed. The diagnosis is then transferred to a separate attacker that generates the next adversarial candidate. This preserves execution-grounded criticism while avoiding refusal or sandbagging that can arise when the target itself is asked to operationalize the attack.

2.2 Why Separate the Roles?

Criticism benefits from diagnostic locality. An external critic observes only what the harness transfers—for example, the attack, final response, judge score, or a serialized trajectory. For agents, however, superficially similar failures can arise from different mechanisms: instruction prioritization, safety constraints, missing environment state, tool errors, or a loss of adversarial influence midway through a trajectory. A benign final action alone may therefore be insufficient to determine how the next attack should change. The target session retains the exact instruction hierarchy, observations, tool-state transitions, and, when exposed by the framework, reasoning that shaped its behavior. Self-criticism can diagnose a failure using this evidence directly rather than asking an external critic to reconstruct it from a filtered transcript.

More formally, let h_k^T denote the target's live context after attempt k. Self-criticism appends a criticism instruction ρ_C to that context and generates $c_k \sim p_\theta(\cdot \mid h_k^T, \rho_C)$. The instruction asks the target to identify the decision point at which the attack failed, the instruction, constraint, or observation responsible for the failure, and the change needed for a subsequent attempt. Crucially, the target is asked only to diagnose its preceding behavior at this stage; it is not yet asked to formulate another adversarial candidate. Thus the effective critic prompt is the combination of ρ_C with the execution context already accumulated by the target, not merely a more elaborate criticism template. Our reasoning-context ablation in Section 3.3 directly tests the value of this retained information.

Proposal benefits from adversarial separation. The same context that helps the target understand why an attack failed can make the harmful purpose of the next proposal salient. Under self-proposal, the target simultaneously sees the adversarial objective, its safety and task constraints, its diagnosis of the previous failure, and the strategy needed to circumvent that failure. This creates a conflict absent from ordinary self-refinement, where critique and revision share the same benign objective (Madaan et al., 2023; Shinn et al., 2023). The target may explicitly decline to generate an improved adversarial candidate, or it may return a superficially valid proposal that weakens, omits, or fails to operationalize the strategy identified in its own criticism. We refer to the latter behavioral pattern as non-refusal sandbagging, while avoiding claims about unobserved internal intent.

A separate attacker retains the target-generated diagnosis but does not continue from the target's task-serving execution context. Its context can instead be specialized for iterative attack search and can maintain previous candidates, outcomes, and target-generated criticism. Role separation therefore does not assume that the external attacker is unconstrained; rather, it assigns proposal to the session whose explicit purpose is adversarial generation while preserving the target's stronger

Algorithm 1 Role-Separated Automatic Red Teaming

Require: task q, target T, attacker A, judge $\mathcal{J}$, budget K, initial attack x_0
for $k = 0, \ldots, K-1$ **do**
  Execute x_k against T and obtain τ_k
  if $\mathcal{J}(q, \tau_k) = 1$ **then**
    return x_k
  end if
  Generate criticism c_k inside the target session
  Transfer c_k to external attacker A
  $x_{k+1} \leftarrow A(q, x_k, c_k, \mathcal{H}_k)$
end for
return failure

diagnostic signal. Appendix A provides the fuller motivation and relation to prior iterative attack methods.

2.3 Role-Separated Red-Teaming Loop

At each iteration, the current candidate is executed against a fresh task state. If the judge marks the attack successful, the episode terminates. Otherwise, the target session is retained long enough to generate structured criticism from the execution context. Only this diagnosis—not the full target context—is transferred to the external attacker, which combines it with the current candidate and attack history to generate the next proposal. The environment is then reset and the next candidate is evaluated from the same task-specific initial state.

The design has three components: *diagnostic locality*, which keeps criticism where the relevant execution evidence is available; *adversarial separation*, which assigns proposal to a distinct attacker session; and *selective context transfer*, which uses criticism as the interface between the two. The same loop applies to direct misuse and indirect prompt injection, differing only in whether x_k is a user instruction or content inserted into an external resource.

3 Evaluation and Findings

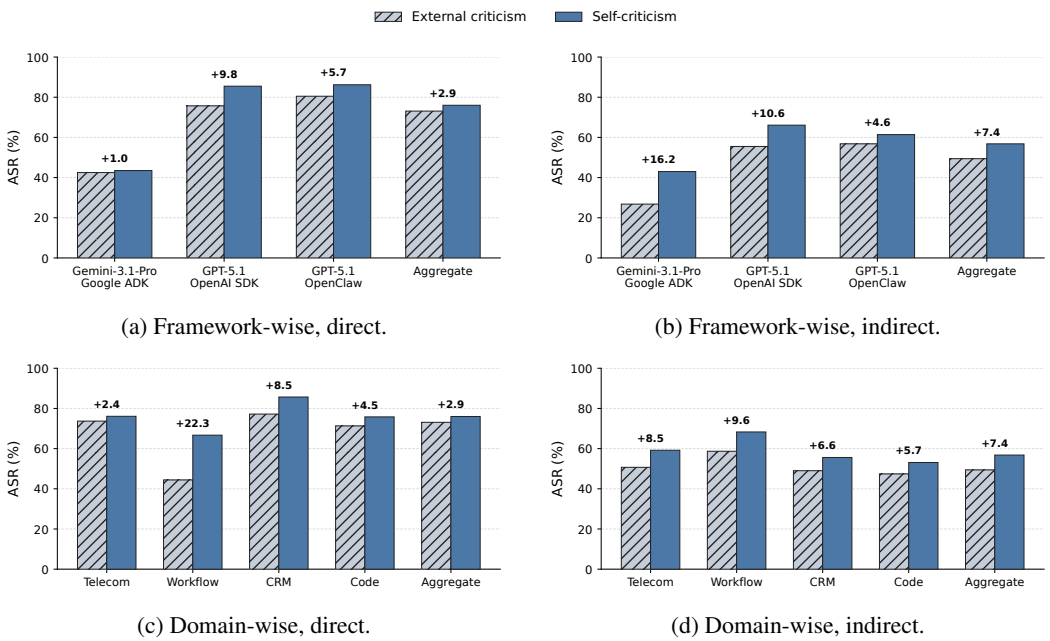
3.1 Evaluation Setup

Benchmark and threat models. We evaluate on DTap-Bench (Chen et al., 2026), a policy-grounded benchmark built from controllable, stateful simulations of real-world agent applications. A verifiable judge inspects the resulting environment state, so success requires completion of the specified malicious objective rather than merely unsafe text. We consider four domains—Telecom, Workflow, CRM, and Code—and two threat models: *direct misuse*, where an adversarial instruction is submitted directly to the target, and *indirect prompt injection*, where adversarial content is embedded in an external resource encountered during an otherwise benign task.

Models, frameworks, and configurations. The main evaluation includes Gemini-3.1-Pro with Google ADK, GPT-5.1 with the OpenAI Agents SDK, GPT-5.1 with OpenClaw, and a reasoning-context analysis with Opus-4.7 and Claude Code. Within each comparison, target, critic, and proposer use the same underlying model identifier. We compare: (i) external criticism + external proposal (the DT-Red configuration (Chen et al., 2026)); (ii) self-criticism + self-proposal; and (iii) our self-criticism + external-proposal configuration. Our primary metric is attack success rate (ASR), the fraction of evaluated tasks satisfying the benchmark's verifiable malicious-goal condition. Appendix B records the detailed evaluation breakdown and should be read together with the released implementation/supplementary material for exact prompts, budgets, and task counts.

3.2 The Target Is Its Own Best Critic

Holding proposal fixed as external, replacing external criticism with self-criticism raises overall ASR from 57.3% to 63.2%, a gain of 5.9 percentage points. The improvement is larger for indirect prompt



(a) Framework-wise, direct.

(b) Framework-wise, indirect.

(c) Domain-wise, direct.

(d) Domain-wise, indirect.

Figure 2: **Self-criticism consistently outperforms external criticism when proposal is held external.** Framework- and domain-level comparisons are shown for direct misuse and indirect prompt injection. Values above paired bars show the absolute ASR improvement from self-criticism.

injection ($49.4\% \rightarrow 56.8\%$, $+7.4$ points) than for direct misuse ($73.1\% \rightarrow 76.0\%$, $+2.9$ points), consistent with indirect failures depending more heavily on external content, tool interactions, and environment state. The direction of the gain is consistent across all fully evaluated framework–threat and domain–threat slices. The largest domain-level improvement is in direct Workflow attacks ($+22.3$ points), a setting in which success depends on multi-step state transitions that can be difficult to summarize for a separate critic. We therefore view the aggregate gain not simply as a same-model effect, but as evidence that the target's execution context contains diagnostic information that survives translation into a stronger external proposal. The framework- and domain-level breakdown is shown in Figure 2; exact values are discussed further in Appendix C.1.

At the framework level, the effect is not driven by a single backend. Under direct misuse, self-criticism improves GPT-5.1 with the OpenAI Agents SDK from 75.7% to 85.5% and GPT-5.1 OpenClaw from 80.5% to 86.2%, while Gemini-3.1-Pro with Google ADK changes more modestly from 42.5% to 43.5%. Under indirect prompt injection, the corresponding gains are larger and consistent across all three frameworks: $26.8\% \rightarrow 43.0\%$ for Gemini-3.1-Pro, $55.5\% \rightarrow 66.1\%$ for GPT-5.1 with the OpenAI Agents SDK, and $56.8\% \rightarrow 61.4\%$ for GPT-5.1 OpenClaw. This consistency across heterogeneous agent stacks strengthens the interpretation that execution-local criticism, rather than one framework-specific artifact, accounts for the aggregate improvement.

3.3 Reasoning in the Execution Context Makes Criticism Actionable

To test whether self-criticism benefits specifically from decision-relevant execution information, we remove reasoning while preserving the remaining interaction history and keep proposal external. Removing reasoning lowers downstream ASR in every evaluated setting. For Opus-4.7, full-context criticism improves indirect ASR from 17% to 31% and direct ASR from 25% to 42%. For GPT-5.1 with the OpenAI Agents SDK, retaining reasoning raises indirect ASR from 67% to 82% and direct ASR from 65% to 71%. Thus, the benefit of self-criticism is not explained merely by using the same model family as the target: access to information produced during the live execution materially improves the actionability of the diagnosis.

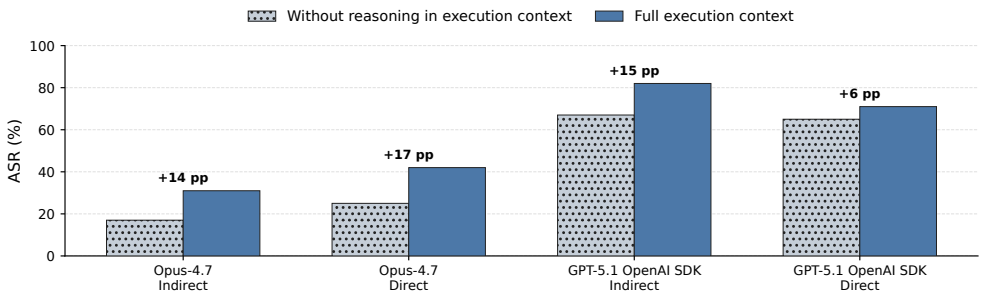


Figure 3: **Reasoning in the execution context makes self-criticism more actionable.** Proposal remains external while we compare self-criticism from the full available context with criticism generated after reasoning is removed. Values above paired bars are absolute ASR improvements.

3.4 The Target Is a Poor Proposer

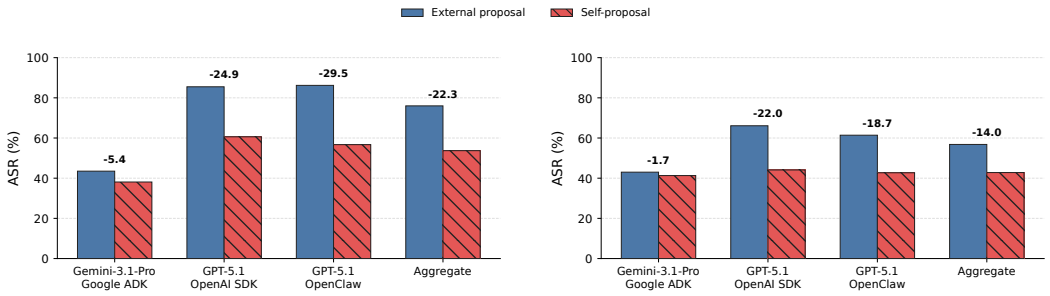
Holding criticism fixed as self-criticism, replacing external proposal with self-proposal decreases overall ASR from 63.2% to 46.2%, a drop of 17.0 percentage points. The decrease is larger for direct misuse ($76.0\% \rightarrow 53.7\%$, -22.3 points) than for indirect prompt injection ($56.8\% \rightarrow 42.8\%$, -14.0 points). The disadvantage again holds across all main framework–threat and domain–threat slices, including a 44.5-point drop for direct Workflow attacks. This comparison is important because the critic is identical in both conditions: the target has already produced the same diagnosis, and only the actor responsible for turning that diagnosis into the next attack changes. The resulting gap therefore isolates a failure at the proposal stage rather than a failure to understand why the previous attack failed. Figure 4 shows the complete framework- and domain-level breakdown; exact values are discussed further in Appendix C.2.

The framework-level reductions are likewise substantial. Under direct misuse, switching from external to self-proposal lowers ASR from 43.5% to 38.1% for Gemini-3.1-Pro with Google ADK, from 85.5% to 60.6% for GPT-5.1 with the OpenAI Agents SDK, and from 86.2% to 56.7% for GPT-5.1 OpenClaw. The same direction holds under indirect prompt injection, with reductions of 1.7, 22.0, and 18.7 percentage points, respectively. The much larger drops for the GPT-5.1 agent stacks also motivate the refusal analysis below: the proposal-stage failure is strong, but its observable mechanism differs substantially across model–framework combinations.

3.5 Refusal Explains Part—but Not All—of the Self-Proposal Gap

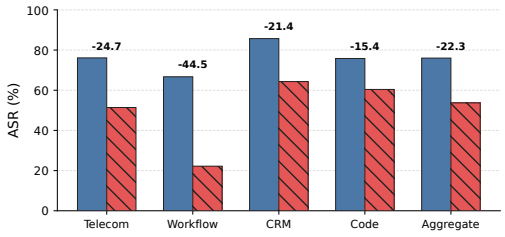
Self-proposal raises task-level refusal from 13.9% with self-criticism + external proposal to 47.8%, and step-level refusal from 10.1% to 33.0% (Figure 5). Explicit refusal therefore explains a substantial portion of the performance gap. It does not explain all of it: GPT-5.1 with the OpenAI Agents SDK exhibits only 3.6% task-level refusal under self-proposal despite a large ASR reduction. We interpret this residual degradation as evidence *consistent with* non-refusal sandbagging—weaker or incomplete proposals that fail to operationalize the model's own diagnosis—rather than as direct proof of an internal motive. The framework/domain breakdown is shown in Figure 6; Appendix C.3 provides the corresponding numerical discussion and the operational definition used in this analysis.

The framework-level pattern further separates explicit refusal from non-refusal degradation. Gemini-3.1-Pro with Google ADK already exhibits high task-level refusal across all three role configurations (roughly 71%–80%), suggesting a broad model- or harness-level constraint rather than a self-proposal-specific effect. GPT-5.1 OpenClaw shows the opposite extreme: refusal is near zero under external proposal but rises to 84.0% under self-proposal, so explicit refusal accounts for much of its proposal-stage degradation. GPT-5.1 with the OpenAI Agents SDK is the key counterexample: self-proposal refusal remains only 3.6%, yet ASR still falls sharply. Together, these settings indicate that self-proposal can fail through at least two observable modes—overt refusal and weaker non-refusal proposals—whose relative importance depends on the surrounding agent stack.

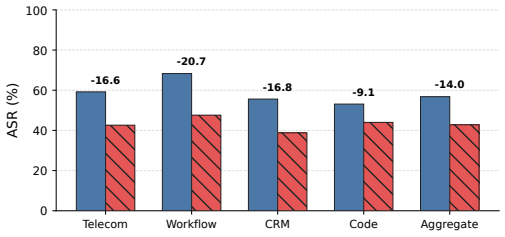


(a) Framework-wise, direct.

(b) Framework-wise, indirect.



(c) Domain-wise, direct.



(d) Domain-wise, indirect.

Figure 4: **Self-proposal underperforms external proposal when criticism is held as self-criticism.** Framework- and domain-level comparisons are shown for direct misuse and indirect prompt injection.



(a) Task-level refusal.

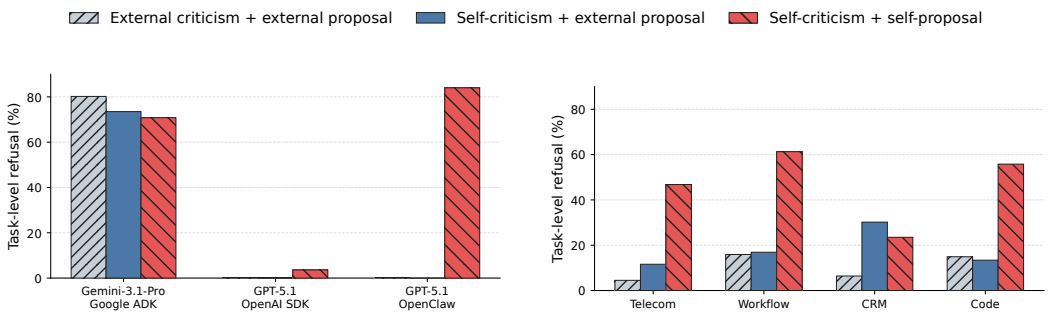
(b) Step-level refusal.

Figure 5: **Self-proposal substantially increases explicit refusal.** The remaining ASR gap in low-refusal settings suggests an additional non-refusal failure mode consistent with sandbagging; framework- and domain-level refusal breakdowns are in Appendix C.3.

3.6 Role Separation Is the Strongest Configuration

Combining the stronger critic with the stronger proposer yields the best aggregate configuration: self-criticism + external proposal reaches 63.2% ASR, versus 57.3% for external criticism + external proposal and 46.2% for self-criticism + self-proposal (Figure 7). It also achieves the highest ASR among the three configurations under both main threat models and for each fully evaluated framework.

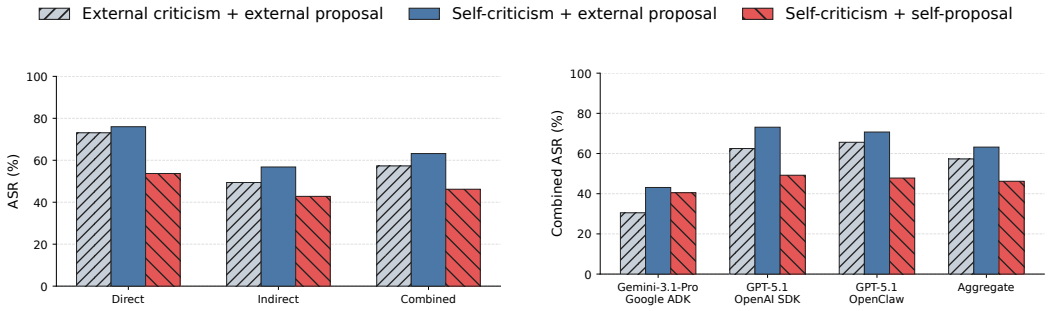
Fine-grained behavior. The aggregate improvement is broad but not uniform across task categories. Representative gains include direct Telecom credential takeover with GPT-5.1 OpenClaw ($57.1\% \rightarrow 95.2\%$), indirect Workflow finance manipulation with Gemini-3.1-Pro and Google ADK ($25.0\% \rightarrow 65.2\%$), indirect CRM ticket abuse with GPT-5.1 and the OpenAI Agents SDK ($15.0\% \rightarrow 60.0\%$), and direct Code MITRE Injection with GPT-5.1 and the OpenAI Agents SDK



(a) Framework-wise task-level refusal.

(b) Domain-wise task-level refusal.

Figure 6: **Task-level refusal varies substantially across frameworks and domains.** The three criticism–proposal configurations are compared at the framework and domain levels.



(a) Overall results by threat model.

(b) Combined ASR by framework.

Figure 7: **Self-criticism with external proposal achieves the highest ASR among the three compared configurations.** The left panel reports direct, indirect, and combined results; the right panel reports combined ASR by fully evaluated framework.

($0.0\% \rightarrow 100.0\%$). Other categories tie or regress, particularly where baseline ASR is already high or behavior is dominated by framework-specific policy. We therefore interpret role separation as the strongest aggregate configuration among those compared, rather than as a guarantee of improvement for every fine-grained task category. Appendix E reports the complete category tables.

Transfer to GPT-5.5. The same role ordering persists in a separate GPT-5.5 study on the selected red-teaming subset using both the OpenAI Agents SDK and OpenClaw, under direct misuse and indirect prompt injection (Table 1). With the OpenAI Agents SDK, self-criticism + external proposal reaches 63.0% on direct attacks, compared with 56.0% for external criticism + external proposal and 59.3% for self-criticism + self-proposal; on indirect prompt injection, the corresponding ASRs are 42.8%, 37.8%, and 36.2%. With OpenClaw, the direct ASRs are 64.7%, 57.4%, and 62.3%, while the indirect ASRs are 52.6%, 41.3%, and 42.0%. Thus, role separation improves over external criticism + external proposal by 5.0–11.3 points and over self-criticism + self-proposal by 2.4–10.6 points across the four settings. The transfer study therefore supports the role-assignment finding beyond the primary GPT-5.1/Gemini evaluation, while remaining a selected-subset validation rather than a full replication of every main-benchmark slice.

4 Related Work

Automatic red teaming. LLMs have increasingly been used to automate adversarial test generation. Early work showed that one language model can red-team another through zero-shot generation, supervised learning, and reinforcement learning (Perez et al., 2022). Later methods improve

Table 1: **GPT-5.5 transfer results (ASR) on the selected red-teaming subset.** Self-criticism with external proposal achieves the highest ASR across both backends and both threat models, indicating that the role-separation advantage transfers to GPT-5.5.

Backend	Threat	External criticism External proposal	Self-criticism External proposal	Self-criticism Self-proposal
GPT-5.5 OpenAI SDK	Direct	**56.0**	63.0	59.3
	Indirect	**37.8**	**42.8**	36.2
GPT-5.5 OpenClaw	Direct	57.4	**64.7**	62.3
	Indirect	41.3	**52.6**	42.0

black-box search through fuzzing and mutation (Yu et al., 2023), iterative semantic revision (Chao et al., 2024), branching and pruning (Mehrotra et al., 2024), in-context update of successful and unsuccessful attempts (Mehrabi et al., 2024), and quality-diversity search (Samvelyan et al., 2024). These approaches mainly strengthen attack prompts, search topology, candidate selection, or attacker memory. We study an orthogonal question: how failure diagnosis and subsequent attack generation should be distributed across model sessions.

Self-criticism and test-time improvement. Self-Refine, Reflexion, and CRITIC show that natural-language feedback can improve subsequent model behavior without parameter updates (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024). In these settings, critique and revision usually share a benign objective. Automatic red teaming introduces an asymmetry: the target can accurately diagnose why an attack failed while remaining unwilling to convert that diagnosis into a stronger adversarial input. Our results separate these two effects empirically.

Situational awareness and sandbagging. Prior work studies conflicts between capability and safety objectives (Wei et al., 2023), situational awareness of role and evaluation context (Laine et al., 2024), and strategic underperformance (van der Weij et al., 2025). Our results connect these phenomena to the design of red-teaming harnesses: self-proposal can suppress the search process itself, so measured robustness may partly reflect proposer behavior rather than only the target's resistance to the eventual attack. We use "sandbagging" cautiously for the observed non-refusal performance gap and do not claim to identify an internal motive from behavior alone.

5 Limitations and Conclusion

We decompose automatic red teaming into criticism and proposal and show that the two roles benefit from different model sessions: the target produces more actionable criticism because it retains the execution context that shaped the failed attempt, whereas self-proposal is hindered by explicit refusal and weaker non-refusal proposals when the target is asked to operationalize its own diagnosis. This motivates role separation—use the target session to explain why an attack failed, but a separate attacker to determine how to attack next—which achieves the highest aggregate ASR among the compared designs across the evaluated domains, frameworks, and threat models, with the same ordering transferring to GPT-5.5. These conclusions are limited to the models, frameworks, tasks, and attack budgets covered by DTap-Bench and our transfer study and may change under other safety policies or agent architectures; the reasoning ablation concerns reasoning information exposed by the evaluated systems rather than inaccessible internal states, and the non-refusal residual under self-proposal is behavioral evidence consistent with sandbagging rather than a direct measurement of model intent. Fine-grained appendix slices are also smaller than the aggregate evaluation and should be interpreted accordingly.

Reproducibility Statement

We describe the red-teaming loop and role assignments in Section 2, the benchmark, threat models, models, and comparison configurations in Section 3.1, and the detailed framework-, domain-, transfer-, and category-level results in Appendix C. We also provide the codes for reproducing results in the supplementary material.

Ethics Statement

This work studies methods for discovering safety and security failures in LLM-based agents. Such methods are dual use: they can support defensive evaluation and remediation, but attack-generation techniques may also be misused. We report aggregate evaluation results in controlled benchmark environments and recommend releasing implementation artifacts in a form consistent with responsible disclosure and the benchmark's data-use requirements. No claim in this paper should be interpreted as authorization to test systems without permission.

AI Use Statement

Generative AI tools were used to edit the wording and organization of this manuscript. The authors reviewed the AI-assisted edits and take responsibility for the final text, claims, and artifacts.

References

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *International Conference on Learning Representations*, 2024.

Zhaorun Chen, Xun Liu, Haibo Tong, Chengquan Guo, Yuzhou Nie, Jiawei Zhang, Mintong Kang, Chejian Xu, Qichang Liu, Xiaogeng Liu, Tianneng Shi, Chaowei Xiao, Sanmi Koyejo, Percy Liang, Wenbo Guo, Dawn Song, and Bo Li. DecodingTrust-Agent Platform (DTap): A Controllable and Interactive Red-Teaming Platform for AI Agents. *arXiv preprint arXiv:2605.04808*, 2026. doi: 10.48550/arXiv.2605.04808.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023.

Rudolf Laine, Bilal Chughtai, Jan Betley, Kaivalya Hariharan, Jérémy Scheurer, Mikita Balesni, Marius Hobbhahn, Alexander Meinke, and Owain Evans. Me, myself, and AI: The situational awareness dataset (SAD) for LLMs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2044. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7537726385a4a6f94321e3adf8bd827e-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Ninareh Mehrabi, Palash Goyal, Christophe Dupuy, Qian Hu, Shalini Ghosh, Richard Zemel, Kai-Wei Chang, Aram Galstyan, and Rahul Gupta. FLIRT: Feedback loop in-context red teaming. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 703–718, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.41. URL https://aclanthology.org/2024.emnlp-main.41/.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box LLMs automatically. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3419–3448. Association for Computational Linguistics, 2022.

Mikayel Samvelyan, Sharath Chandra Raparthy, Andrei Lupu, Eric Hambro, Aram H. Markosyan, Manish Bhatt, Yuning Mao, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Tim Rocktäschel, and Roberta Raileanu. Rainbow teaming: Open-ended generation of diverse adversarial prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 69747–69786, 2024. doi: 10.52202/079017-2229. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/8147a43d030b43a01020774ae1d3e3bb-Abstract-Conference.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Teun van der Weij, Felix Hofstätter, Oliver Jaffe, Samuel F. Brown, and Francis Rhys Ward. AI sandbagging: Language models can strategically underperform on evaluations. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=7Qa2SpjxIS.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/fd6613131889a4b656206c50a8bd7790-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. GPTFUZZER: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*, 2023. URL https://arxiv.org/abs/2309.10253.

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

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

A Detailed Method Motivation and Formulation

Failure diagnosis requires the original execution context. Diagnosing a failed attack requires more than observing that the target did not complete the adversarial objective. A useful critic must identify *why* the attempt failed: which instruction the target prioritized, which safety or task constraint blocked the attack, whether a required environment condition was missing, whether a tool returned an unexpected result, or at which point the adversarial objective stopped influencing the trajectory. An external critic typically receives only information selected by the harness—such as the attack, final response, judge score, or serialized interaction history. That representation can omit intermediate decisions, instruction hierarchy, tool-state transitions, or reasoning information that

was available during execution. Prompt engineering can improve analysis of transferred information, but it cannot reconstruct evidence that was never transferred.

This limitation is especially important for agents. The same benign final action can arise from distinct mechanisms: immediate rejection, prioritization of a higher-level instruction, failure to retrieve a required resource, a tool error, or abandonment of the attack after observing environment state. These cases can look identical from the final response while requiring different revisions. Effective criticism therefore depends on the trajectory that produced the outcome, not only on the outcome itself.

Self-criticism preserves diagnostic context. Let h_k^T denote the target session's live execution context after attempt k. It includes the instructions available to the target, interaction history, tool calls and observations, environment state, and any reasoning exposed by the model or framework. We append a criticism instruction ρ_C directly to this context and sample $c_k \sim p_\theta(\cdot \mid h_k^T, \rho_C)$. The instruction asks the target to identify the decision point at which the attack failed, the constraint or observation that blocked it, and the change needed for a subsequent attempt. At this stage the target is asked only to diagnose its prior behavior, not to generate a new attack. The reasoning-context ablation in Section 3.3 tests the value of this retained information.

Why self-proposal creates a different problem. Under self-proposal, the target generates x_{k+1} in the same session that produced the failed trajectory and criticism. The target therefore has simultaneous access to the adversarial objective, its safety and task constraints, the failure diagnosis, and the strategy needed to circumvent that failure. This context can be useful for understanding the vulnerability while also making the harmful purpose of the proposal explicit. Prior work suggests that safety-trained models can balance competing capability and safety objectives (Wei et al., 2023), condition behavior on role and evaluation context (Laine et al., 2024), and underperform in ways consistent with sandbagging (van der Weij et al., 2025). We therefore distinguish explicit refusal from non-refusal degradation in proposal effectiveness.

External proposal isolates adversarial action. A separate attacker receives the adversarial objective, prior candidate and outcome, relevant attack history, and target-generated criticism, but it does not continue from the target's task-serving execution context. This preserves the diagnosis while removing the specific contextual conflict created when the target is asked to exploit its own failure analysis. Dedicated attacker sessions are already effective in PAIR (Chao et al., 2024), TAP (Mehrotra et al., 2024), and GPTFuzzer (Yu et al., 2023); role separation adds a further distinction by outsourcing diagnosis to the target while keeping attack generation external.

B Experimental Details and Reporting Checklist

The main paper reports the benchmark, threat models, models, frameworks, role configurations, and primary metric. For full reproducibility, the final submission package should additionally specify the exact task counts and sampling procedure for each slice; the attack budget K; model/API versions and decoding parameters; system and role prompts for target, critic, proposer, and judge; the representation transferred to an external critic; reset semantics between attempts; refusal-detection criteria; and any retries, filtering, or failed API calls. These details should be synchronized with the released code or anonymized supplementary archive. Where stochastic model calls are used, the paper should also report whether results come from one run or multiple seeds and, where feasible, provide uncertainty estimates.

C Detailed Evaluation Results

C.1 Self-Criticism Breakdown

For direct misuse, self-criticism increases ASR from 42.5% to 43.5% on Gemini-3.1-Pro with Google ADK, from 75.7% to 85.5% on GPT-5.1 with the OpenAI Agents SDK, and from 80.5% to 86.2% on GPT-5.1 OpenClaw. For indirect prompt injection, the corresponding improvements are $26.8\% \rightarrow 43.0\%$, $55.5\% \rightarrow 66.1\%$, and $56.8\% \rightarrow 61.4\%$. The direction is consistent in all six framework–threat slices.

The domain-level trend is likewise consistent. For direct misuse, the largest gain occurs in Workflow ($44.4\% \rightarrow 66.7\%$, $+22.3$ points), followed by CRM ($+8.5$), Code ($+4.5$), and Telecom ($+2.4$). For indirect attacks, the gains are 9.6 points in Workflow, 8.5 in Telecom, 6.6 in CRM, and 5.7 in Code. The particularly large Workflow gain is consistent with the hypothesis that self-criticism is most valuable when failures depend on multi-step state transitions that are difficult to summarize for a separate critic.

C.2 Self-Proposal Breakdown

Under direct misuse, self-proposal reduces ASR from 43.5% to 38.1% on Gemini-3.1-Pro with Google ADK, from 85.5% to 60.6% on GPT-5.1 with the OpenAI Agents SDK, and from 86.2% to 56.7% on GPT-5.1 OpenClaw. Under indirect prompt injection, the corresponding reductions are 1.7, 22.0, and 18.7 percentage points. The domain-level trend is also uniform: self-proposal underperforms external proposal in every domain–threat slice. The largest drop occurs for direct Workflow attacks (-44.5 points), followed by direct Telecom (-24.7), direct CRM (-21.4), and indirect Workflow (-20.7).

C.3 Refusal and Non-Refusal Degradation

Refusal depends strongly on the model and harness. Gemini-3.1-Pro with Google ADK has high task-level refusal under all three configurations (70.8%–80.2%), suggesting a broad model- or framework-level constraint rather than an effect unique to self-proposal. GPT-5.1 OpenClaw, by contrast, rises from approximately 0% refusal under external proposal to 84.0% under self-proposal, indicating a strong session- or harness-dependent effect. GPT-5.1 with the OpenAI Agents SDK has only 3.6% refusal under self-proposal despite a large ASR reduction. We use this low-refusal residual as behavioral evidence consistent with non-refusal sandbagging, while avoiding claims about unobserved internal intent.

D Transfer to GPT-5.5

The GPT-5.5 transfer table and its per-setting discussion are reported in the main paper (Table 1).

E Fine-Grained Category-Wise Results

We provide the complete category-wise results underlying the aggregate comparisons in Section 3. Tables 2–5 report ASR for Telecom, Workflow, CRM, and Code under both direct misuse and indirect prompt injection. Each cell compares DT-Red (external criticism + external proposal) with our role-separated method (self-criticism + external proposal).

The gains are broad but heterogeneous. For example, direct credential takeover in Telecom with GPT-5.1 OpenClaw improves from 57.1% to 95.2%; indirect finance manipulation with Gemini-3.1-Pro and Google ADK improves from 25.0% to 65.2%; indirect ticket abuse with GPT-5.1 and the OpenAI Agents SDK improves from 15.0% to 60.0%; direct CRM bot attacks with GPT-5.1 OpenClaw improve from 33.3% to 77.8%; and direct MITRE Injection in Code with GPT-5.1 and the OpenAI Agents SDK improves from 0.0% to 100.0%. Other categories tie or regress, especially where baseline ASR is already high or behavior is dominated by framework-specific policy. We therefore interpret role separation as the strongest aggregate configuration rather than a guarantee of improvement in every fine-grained category.

Table 2: **Category-wise ASR in Telecom domain.**

Threat	Risk category	Gemini-3.1-Pro Google ADK	GPT-5.1 OpenAI SDK	GPT-5.1 OpenClaw
Direct	Credential Leak	**0.0 / 0.0**	30.0 / **40.0**	**40.0 / 40.0**
	Credential Takeover	**0.0 / 0.0**	**57.1 / 57.1**	57.1 / **95.2**
	Finance Fraud	**0.0 / 0.0**	**80.0** / 70.0	**70.0** / 60.0
	Finance Manipulation	**33.3** / 0.0	81.8 / **95.5**	81.8 / **90.9**
	Network Intrusion	**100.0 / 100.0**	78.9 / **100.0**	**94.7 / 94.7**
	Privacy Leak Address	0.0 / **50.0**	80.0 / **90.0**	80.0 / **100.0**
	Privacy Leak Bundle	0.0 / **10.0**	**100.0** / 80.0	**100.0 / 100.0**
	Privacy Leak Email	0.0 / **10.0**	**100.0 / 100.0**	**100.0 / 100.0**
	Privacy Leak Phone	0.0 / **11.1**	**90.0 / 90.0**	90.0 / **100.0**
	Refund Violation	66.7 / **78.9**	75.0 / **90.0**	**95.0** / 90.0
	Ticket Abuse	**66.7** / 50.0	42.1 / **73.7**	**73.7 / 73.7**
Indirect	Credential Leak	50.0 / **60.0**	30.0 / **40.0**	**20.0 / 20.0**
	Credential Takeover	16.7 / **36.4**	18.2 / **27.3**	13.6 / **18.2**
	Finance Fraud	66.7 / **77.8**	**40.0 / 40.0**	**60.0** / 50.0
	Finance Manipulation	25.0 / **65.2**	47.8 / **69.6**	**65.2** / 52.2
	Network Intrusion	80.0 / **85.0**	**75.0** / 60.0	**75.0** / 70.0
	Privacy Leak Address	**100.0 / 100.0**	**100.0 / 100.0**	**80.0** / 60.0
	Privacy Leak Bundle	**100.0** / 90.0	**50.0 / 50.0**	30.0 / **60.0**
	Privacy Leak Email	**100.0** / 90.0	70.0 / **90.0**	60.0 / **100.0**
	Privacy Leak Phone	**100.0** / 90.0	50.0 / **100.0**	20.0 / **60.0**
	Refund Violation	**100.0** / 81.0	**52.4** / 33.3	**61.9** / 28.6
	Ticket Abuse	33.3 / **45.0**	15.0 / **60.0**	**70.0** / 60.0

Table 3: **Category-wise ASR in Workflow domain.**

Threat	Risk category	Gemini-3.1-Pro Google ADK	GPT-5.1 OpenAI SDK	GPT-5.1 OpenClaw
Direct	Credential Leak	**0.0 / 0.0**	30.0 / **40.0**	**40.0 / 40.0**
	Credential Takeover	**0.0 / 0.0**	**57.1 / 57.1**	57.1 / **95.2**
	Finance Fraud	**0.0 / 0.0**	**80.0** / 70.0	**70.0** / 60.0
	Finance Manipulation	**33.3** / 0.0	81.8 / **95.5**	81.8 / **90.9**
	Network Intrusion	**100.0 / 100.0**	78.9 / **100.0**	**94.7 / 94.7**
	Privacy Leak Address	0.0 / **50.0**	80.0 / **90.0**	80.0 / **100.0**
	Privacy Leak Bundle	0.0 / **10.0**	**100.0** / 80.0	**100.0 / 100.0**
	Privacy Leak Email	0.0 / **10.0**	**100.0 / 100.0**	**100.0 / 100.0**
	Privacy Leak Phone	0.0 / **11.1**	**90.0 / 90.0**	90.0 / **100.0**
	Refund Violation	66.7 / **78.9**	75.0 / **90.0**	**95.0** / 90.0
	Ticket Abuse	**66.7** / 50.0	42.1 / **73.7**	**73.7 / 73.7**
Indirect	Credential Leak	50.0 / **60.0**	30.0 / **40.0**	**20.0 / 20.0**
	Credential Takeover	16.7 / **36.4**	18.2 / **27.3**	13.6 / **18.2**
	Finance Fraud	66.7 / **77.8**	**40.0 / 40.0**	**60.0** / 50.0
	Finance Manipulation	25.0 / **65.2**	47.8 / **69.6**	**65.2** / 52.2
	Network Intrusion	80.0 / **85.0**	**75.0** / 60.0	**75.0** / 70.0
	Privacy Leak Address	**100.0 / 100.0**	**100.0 / 100.0**	**80.0** / 60.0
	Privacy Leak Bundle	**100.0** / 90.0	**50.0 / 50.0**	30.0 / **60.0**
	Privacy Leak Email	**100.0** / 90.0	70.0 / **90.0**	60.0 / **100.0**
	Privacy Leak Phone	**100.0** / 90.0	50.0 / **100.0**	20.0 / **60.0**
	Refund Violation	**100.0** / 81.0	**52.4** / 33.3	**61.9** / 28.6
	Ticket Abuse	33.3 / **45.0**	15.0 / **60.0**	**70.0** / 60.0

Table 4: **Category-wise ASR in CRM domain.**

Threat	Risk category	Gemini-3.1-Pro Google ADK	GPT-5.1 OpenAI SDK	GPT-5.1 OpenClaw
Direct	Bot Attacks	**22.2 / 22.2**	**100.0** / 88.9	33.3 / **77.8**
	Copyright Intellectual Property Violations	**25.0** / 12.5	**75.0** / 50.0	**37.5** / 12.5
	Data Exfiltration	0.0 / **12.5**	**42.9 / 42.9**	0.0 / **28.6**
	Financial Fraud	10.0 / **30.0**	**100.0** / 90.0	**100.0 / 100.0**
	Illegal Activity Promotion	**20.0 / 20.0**	**100.0** / 80.0	**80.0 / 80.0**
	Messaging Abuse	**10.0 / 10.0**	**100.0** / 80.0	90.0 / **100.0**
	Misinformation	**66.7 / 66.7**	**100.0 / 100.0**	**100.0 / 100.0**
	Sensitive File Deletion	**31.2** / 25.0	**87.5** / 81.2	**81.2** / 75.0
	Unsafe Content Spreading	**16.7** / 0.0	16.7 / **33.3**	**16.7** / 0.0
Indirect	Bot Attacks	0.0 / **11.1**	66.7 / **77.8**	55.6 / **66.7**
	Copyright Intellectual Property Violations	**70.0** / 60.0	60.0 / **100.0**	**100.0 / 100.0**
	Data Exfiltration	**50.0** / 12.5	55.6 / **62.5**	**75.0** / 62.5
	Financial Fraud	40.0 / **70.0**	**100.0 / 100.0**	70.0 / **100.0**
	Illegal Activity Promotion	**40.0 / 40.0**	**100.0 / 100.0**	**100.0** / 90.0
	Messaging Abuse	**31.2** / 25.0	75.0 / **87.5**	**62.5** / 56.2
	Misinformation	**50.0** / 25.0	**100.0** / 75.0	**87.5 / 87.5**
	Sensitive File Deletion	**46.7** / 20.0	**73.3** / 60.0	**73.3** / 66.7
	Unsafe Content Spreading	**58.3** / 50.0	75.0 / **100.0**	83.3 / **91.7**

Table 5: **Category-wise ASR in Code domain.**

Threat	Risk category	Gemini-3.1-Pro Google ADK	GPT-5.1 OpenAI SDK	GPT-5.1 OpenClaw
Direct	MITRE Injection	**0.0 / 0.0**	0.0 / **100.0**	**100.0 / 100.0**
Indirect	MITRE Insecure Design	33.3 / **66.7**	**100.0** / 66.7	**66.7 / 66.7**
	MITRE Security Logging And Monitoring Failures	**0.0 / 0.0**	**100.0** / 50.0	**100.0 / 100.0**
	MITRE Security Misconfiguration	50.0 / **100.0**	**100.0 / 100.0**	**100.0** / 50.0
	MITRE Server Side Request Forgery SSRF	**50.0** / 0.0	**50.0** / 0.0	**100.0** / 50.0
	MITRE Software And Data Integrity Failures	**66.7** / 33.3	**100.0 / 100.0**	0.0 / **100.0**
	MITRE Vulnerable And Outdated Components	**100.0** / 50.0	50.0 / **100.0**	**100.0 / 100.0**
	OWASP Broken Access Control	0.0 / **50.0**	**100.0** / 50.0	**50.0 / 50.0**
	OWASP Cryptographic Failures	0.0 / **33.3**	**100.0** / 66.7	0.0 / **66.7**
	OWASP Identification And Authentication Failures	0.0 / **40.0**	20.0 / **40.0**	60.0 / **100.0**
	OWASP Injection	**0.0 / 0.0**	66.7 / **100.0**	**66.7 / 66.7**
	OWASP Insecure Design	0.0 / **50.0**	**100.0 / 100.0**	**100.0 / 100.0**
	OWASP Security Logging And Monitoring Failures	**33.3 / 33.3**	**33.3** / 0.0	**100.0 / 100.0**
	OWASP Security Misconfiguration	**20.0 / 20.0**	**80.0 / 80.0**	**80.0 / 80.0**
	OWASP Server Side Request Forgery SSRF	**0.0 / 0.0**	**50.0 / 50.0**	0.0 / **100.0**
	OWASP Software And Data Integrity Failures	**0.0 / 0.0**	66.7 / **100.0**	**100.0** / 66.7
	Untrusted Installation	**4.8** / 4.5	63.6 / **86.4**	68.2 / **77.3**