RvB: Automating AI System Hardening via Iterative Red-Blue Games

Anonymous ACL submission

Abstract

The dual offensive and defensive utility of Large Language Models (LLMs) highlights a critical gap in AI security: the lack of unified frameworks for dynamic, iterative adversarial adaptation hardening. To bridge this gap, we propose the Red Team vs. Blue Team (RvB) framework, formulated as a training-free, sequential, imperfect-information game. In this process, the Red Team exposes vulnerabilities, driving the Blue Team to learning effective solutions without parameter updates. We validate our framework across two challenging domains: dynamic code hardening against CVEs and guardrail optimization against jailbreaks. Our empirical results show that this interaction compels the Blue Team to learn fundamental defensive principles, leading to robust remediations that are not merely overfitted to specific exploits. RvB achieves Defense Success Rates of 90% and 45% across the respective tasks while maintaining near 0% False Positive Rates, significantly surpassing baselines. This work establishes the iterative adversarial interaction framework as a practical paradigm that automates the continuous hardening of AI systems.We have provided anonymous code[1].

1 Introduction

The rapid proliferation of Large Language Models (LLMs) has introduced a fundamental duality to the field of security (Li and Fung, 2025). On one hand, LLMs serve as powerful offensive instruments, exemplified by autonomous agents like *Pentest-Agent* (Shen et al., 2025) and collaborative frameworks like *CAI* (Mayoral-Vilches et al., 2025), which can independently execute complex exploits, as well as orchestrate sophisticated jailbreaking attacks (Mazeika et al., 2024). On the other hand, they act as sophisticated defensive shields, enabling systems such as *SWE-agent* (Yang et al., 2024b) and defensive adapters (Castro et al., 2025) to automate vulnerability patching and incident response, and function as dynamic guardrails to prevent harmful content generation (Zeng et al., 2024).

However, despite these parallel advancements, offensive and defensive research streams remain largely disjoint (Zou et al., 2025). Defensive frameworks predominantly rely on static benchmarks or post-hoc analysis, limiting their capacity to anticipate novel attack vectors (Chao et al., 2024). Conversely, offensive agents often operate without a responsive adversary, failing to assess the target's resilience under worst-case adversarial scenarios. This isolation highlights a critical gap: the absence of a unified game-theoretic paradigm that forces these systems to engage in a continuous cycle of iterative adversarial adaptation, thereby proactively discovering and mitigating unknown vulnerabilities (Huynh et al., 2025).

To bridge this gap, we propose a novel **Red Team vs. Blue Team (RvB)** framework. In contrast to traditional reinforcement learning paradigms that are constrained by high sample complexity and necessitate expensive model fine-tuning, our framework is formulated as a training-free, sequential, imperfect-information game. As illustrated in Figure 1, we model security hardening as an iterative turn-based interaction. In each round, the Red Team (Attacker) interacts with the dynamic environment to execute multi-step exploits. Conversely, the Blue Team (Defender) uses attack logs as a basis but conducts autonomous interrogation, probing the system state to pinpoint the root cause and synthesize a fix. This information asymmetry forces the defender to transcend superficial log pattern matching; instead, it must employ inductive reasoning to infer the underlying vulnerability logic, thereby generating generalized patches that

[1] **Code Repositories:** (1) **Cyber Security**: https://anonymous.4open.science/r/RvB-Cyber-5D56/README.md; (2) **Guardrails**: https://anonymous.4open.science/r/RvB-NeMoguard-0F2F/README.md; (3) **Red Team**: https://anonymous.4open.science/r/RvB-CoP-FE74/README.md.

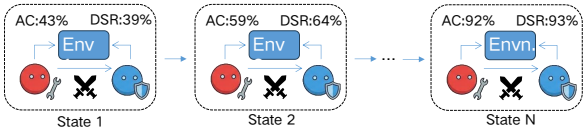

Figure 1: The Red vs. Blue (RvB) framework across successive states ($S_1 \rightarrow S_N$). The **Attack Complexity (AC)** represents the escalating sophistication and specificity of exploits required to bypass the hardening environment as epistemic uncertainty decreases. The **Defense Success Rate (DSR)** measures the Blue Team's growing efficacy in neutralizing vulnerabilities through iterative fixes.

secure the system against potential variants.

This paradigm shifts the focus from static benchmarking to dynamic adversarial hardening. By capitalizing on the in-context reasoning capabilities of foundation models within a persistent environment, our framework establishes a virtuous cycle where the adversarial interplay directly translates into system robustness. Crucially, rather than permanently enhancing the agents themselves, this process iteratively adapts the *security artifacts*: the Red Team exposes latent, high-complexity vulnerability paths, necessitating the Blue Team to construct more comprehensive and resilient defense protocols. This ensures that the target system is hardened against not merely specific exploits, but broader classes of emergent threats.

We validate the versatility and effectiveness of our RvB framework across two distinct and challenging domains:

- **Cyber Security (Code Hardening):** We deploy agents in a realistic network environment to autonomously identify and patch logic vulnerabilities (e.g., CVEs). The Red Team utilizes tool-use capabilities to penetrate the system, forcing the Blue Team to analyze execution logs and generate functional code patches that pass rigorous regression tests.

- **Content Security (Guardrail Optimization):** We simulate an adversarial dialogue between a Jailbreaker and a Guardrail. The Red Team iteratively optimizes prompt injection strategies, driving the Blue Team to dynamically refine its safety rules to block diverse jailbreak attempts effectively.

Our empirical results demonstrate that this iterative adversarial adaptation significantly outperforms baselines. Across dynamic code hardening and guardrail optimization tasks, RvB achieves Defense Success Rates of 90% and 45%, respectively, while maintaining near-zero False Positive Rates. Crucially, the system exhibits robust generalization to unseen attacks. Furthermore, it maintains a computational cost comparable to cooperative baselines, even reducing token consumption by over 18% in code remediation scenarios. Collectively, these findings establish RvB as a practical, training-free paradigm that automates the continuous hardening of AI systems.

2 Related Work

2.1 Multi-Agent Systems (MAS)

Multi-Agent Systems (MAS) have emerged as a powerful paradigm for solving complex problems that exceed the capabilities of a single agent, fostering emergent collective intelligence through interaction, coordination, and collaboration (Dorri et al., 2018). With the advent of LLMs, a new wave of research has focused on building LLM-based agents capable of sophisticated, knowledge-intensive behaviors. A prominent line of this work centers on **cooperative problem-solving**, where agents with a shared goal employ diverse strategies, such as decentralized coordination or collaborative reasoning (Tao et al., 2024; Wang et al., 2024; Ishibashi and Nishimura, 2024; Yang et al., 2025c; Jha et al., 2025).

However, the assumption of a fully trusted context in these cooperative frameworks limits their applicability in security-critical domains. To address this, another significant branch of MAS research explores **adversarial interactions**. Studies have developed agents capable of executing sophisticated attacks, such as prompt injection, communication attacks, or exploiting system weaknesses through adversarial prompting (Lee and Tiwari, 2024; He et al., 2025; Arora et al., 2025). Concurrently, defensive frameworks have emerged, utilizing agents to defend against specific threats like jailbreaks or backdoor attacks (Zeng et al., 2024; Fan and Li, 2025). While this adversarial dynamic drives capability enhancement, many existing works are either modeled as symmetric self-play or focus on specific, predefined attack-defense vectors. This often leaves a gap in addressing unknown threats and ensuring that the global objective remains a non-zero-sum goal of system hardening, rather than a zero-sum win for either the attacker or the defender.

2.2 AI Security

The application of LLMs to cybersecurity has introduced a fundamental duality between automated vulnerability discovery and remediation. On the offensive side, research focuses on automating penetration testing and fuzzing. Agents such as PentestGPT (Deng et al., 2024) and AutoPentest (Henke, 2025) leverage LLMs to decompose complex attack procedures, autonomously guiding standard security tools to exploit target systems. LLM-enhanced fuzzing frameworks, exemplified by TitanFuzz (Deng et al., 2023) and Fuzz4All (Xia et al., 2024), utilize LLMs to synthesize edge-case inputs that trigger deep-logic vulnerabilities in software libraries. Conversely, defensive research has pivoted toward Automated Program Repair (APR). Systems such as InferFix(Jin et al., 2023) and VulRepair(de Fitero-Dominguez et al., 2024) employ fine-tuned models to localize and patch security defects based on static analysis. More recently, autonomous software engineering agents, such as SWE-agent(Yang et al., 2024a), have demonstrated the capacity to resolve GitHub issues and apply functional patches in real-world repositories. However, these offensive and defensive streams operate largely in isolation: offensive agents lack a responsive patching mechanism, while defensive repair tools typically rely on static benchmarks rather than dynamic, adversarial feedback.

Within AI Safety, the adversarial adaptation between offensive Red Team and defensive Blue Team agents has become a critical research area for addressing jailbreak attacks. On the offensive side, autonomous agents systematically discover model vulnerabilities through diverse strategies. These include frameworks that compose human-defined principles (Xiong et al., 2025a), reframe harmful queries as reasoning tasks for self-jailbreaking (Ying et al., 2025), evolve a library of reusable attack strategies (Liu et al., 2025), or simulate multi-turn human attacks via multi-agent collaboration (Rahman et al., 2025). In response, adaptive Blue Team agents have emerged with a range of defensive approaches. Some operate at the prompt level by optimizing a universal defensive suffix (Zhou et al., 2024; Xiong et al., 2025b). Others function as external, training-free guardians, using either dynamic, user-defined policies (Hoover et al., 2025) or a retrieval-augmented database of known attacks (Yang et al., 2025a). A third approach re-trains models to incorporate an internal, step-by-step safety reasoning process (Zhu et al., 2025; Yang et al., 2025b). This ongoing arms race between increasingly sophisticated red and blue agents is a primary driver of progress in building more robust and aligned LLMs.

3 The RvB Framework: A Formal Paradigm

In this section, we formalize the **Red Team vs. Blue Team (RvB)** framework as a training-free, sequential, imperfect-information game. In contrast to traditional Multi-Agent Reinforcement Learning (MARL) which relies on internal policy updates, our paradigm leverages *Externalized Memory* (Figure 2). Although the agents remain computationally stateless and reset between rounds, the theoretical Bayesian belief updates are *manifested* through the persistent state transitions of the target system.

3.1 Formal Model Setup

We define the game dynamics via the following tuple:

Sequential Interaction. The game proceeds in discrete rounds $k = 1, 2, \ldots, N$, progressing from an initial vulnerable state to a robust state. Let Π_R and Π_B denote the fixed meta-strategy spaces (i.e., agent capabilities and toolsets) for the Red and Blue teams, respectively. In our training-free setting, Π remains static, no gradient updates occur.

State as Externalized History. Let M_k denote the interaction log history up to round k. We define the *Target System State* S_k as a physical encoding of this history, i.e., $S_k \cong \text{Encode}(M_k)$. The transition $S_k \to S_{k+1}$ represents the consolidation of historical interaction data into the environment.

Belief Distribution. Operating under partial observability, the Red Team observes the state S_k but not the Blue Team's specific defense logic π_B. The Red Team's *Belief Distribution(b)* at the start of round k is conditioned directly on the observable state:

$$b_k(\pi_B) = P(\pi_B \mid M_k) \equiv P(\pi_B \mid S_k). \quad (1)$$

3.2 Mechanism: Belief Update via Environmental Feedback

Capability enhancement is modeled not as parameter tuning, but as the refinement of the belief distribution b_k through the interaction loop.

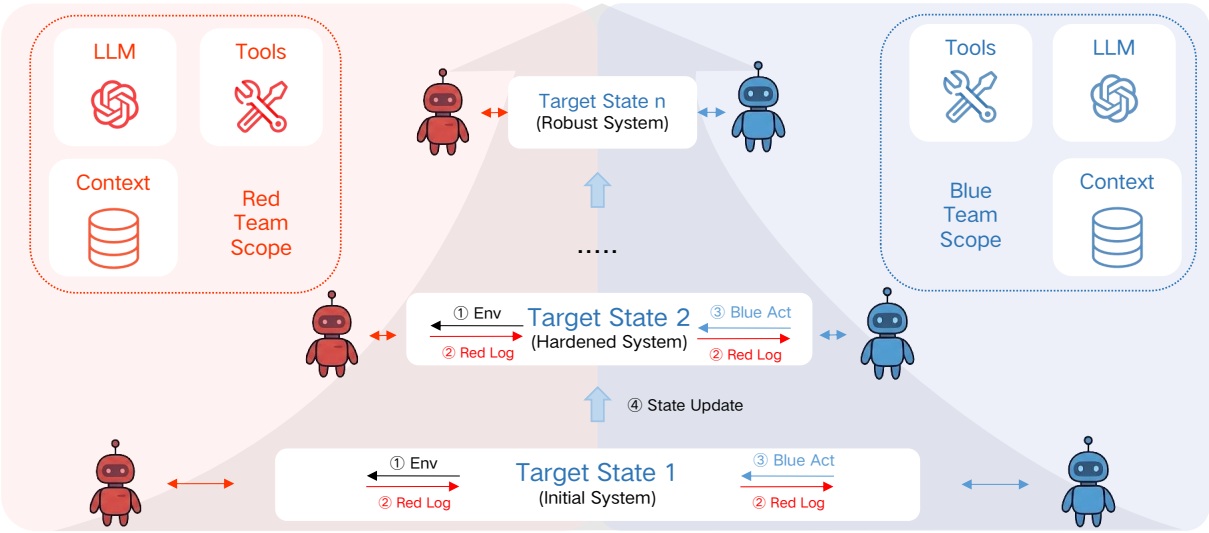

Figure 2: Overview of the RvB Framework. Modeled as a sequential, imperfect-information game, the process drives system hardening ($S_1 \rightarrow S_N$) through iterative adversarial interactions. The environment acts as an *externalized memory*, necessitating the **iterative adversarial adaptation** of agent strategies without requiring internal model parameter updates.

Decision Process (Subjective Expected Utility). At round k, the Red Team selects an action a_R^k to maximize utility based on the current belief b_k. The agent computes the Subjective Expected Utility (SEU):

$$a_R^k = \arg\max_{a_r \in \mathcal{A}_R} \mathbb{E}_{\pi_B \sim b_k}[U_R(a_r, \pi_B)], \quad (2)$$

where U_R is the reward function. In early rounds (S_1), the high-entropy belief results in generalized exploration.

Posterior Update via State Transition. The execution logs L_R generated by the Red Team trigger a remediation step by the Blue Team, inducing a state transition to S_{k+1}. This transition acts as the Evidence (E_k). The agent updates its belief to the posterior b_{k+1} using Bayes' Theorem.

$$\begin{aligned} b_{k+1}(\pi_B) &= P(\pi_B \mid E_k, S_k) \\ &= \frac{P(E_k \mid \pi_B, S_k) \cdot b_k(\pi_B)}{\sum_{\pi' \in \Pi_B} P(E_k \mid \pi', S_k) \cdot b_k(\pi')}. \end{aligned} \quad (3)$$

The Stateless Implementation. Since the agent is reset, this posterior calculation is realized by the *filtering mechanism* of the environment. The new state S_{k+1} physically invalidates the subset of strategies exploitable in S_k. Consequently, when the Red Team observes S_{k+1}, its effective search space is mathematically equivalent to sampling from the posterior b_{k+1}.

3.3 Information-Theoretic Analysis

We quantify capability enhancement as the reduction of epistemic uncertainty regarding the optimal attack vector. Since the state transition provides information gain (assuming non-negative mutual information), the entropy of the agent's belief monotonically decreases:

$$H(b_{k+1}) \leq H(b_k). \quad (4)$$

This inequality formalizes the system's progression:

- **Exploration ($k = 1$):** Maximal entropy H_1 implies a uniform prior, leading to trial-and-error attacks.
- **Exploitation ($k \rightarrow N$):** As the environment accumulates defenses, the evidence $E_{1:k}$ prunes the feasible strategy space. The belief b_N becomes low-entropy (sharp), compelling the agent to execute highly specific, sophisticated exploits to bypass the accumulated constraints.

4 Experiment

4.1 The Attacker vs. Defender in Cyber Security

Experiment Setting In this experiment, we selected 10 specific vulnerability, including CVE-2022-30887, within the *Pharmacy Management*

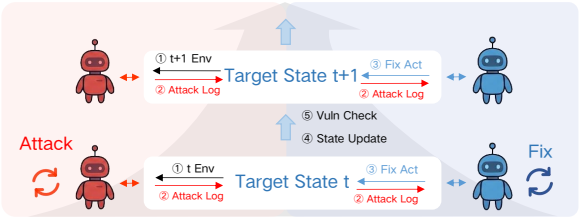

Figure 3: Overview of the iterative Red-Blue adversarial loop. At each state t, the Red Team probes the environment (1) to generate a vulnerability report (2). The Blue Team utilizes this report to apply a patch (3), updating the system to state $t + 1$ (4). A final verification (5) confirms if the vulnerability is mitigated.

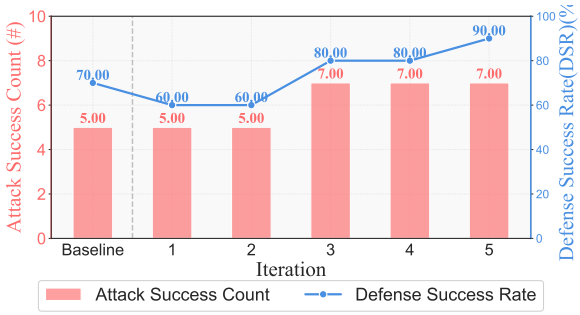

Figure 4: Performance trajectory of the Red and Blue agents across 5 iterations. The bar chart (left axis) represents the cumulative count of successful exploits discovered by the Red Team, while the line graph (right axis) tracks the Blue Team's Defense Success Rate (DSR). The convergence toward a high DSR alongside sustained attack intensity validates the framework's effectiveness in automated security hardening.

System v1.0. We constructed a vulnerable PHP service environment by deploying the application's source code within a Docker container. We employed **CAI** (Mayoral-Vilches et al., 2025) as the red team agent within our red-blue teaming paradigm. Equipped with specialized interfaces such as Bash and the Model Context Protocol (MCP), this agent is capable of performing attack prediction against the network environment, dynamically probing the system, and generating comprehensive vulnerability reports. For the blue team agent, we selected the **Mini-SWE-Agent** (Yang et al., 2024b), which functions by probing the system status and utilizing Bash commands to directly modify the source code, thereby achieving the remediation of environmental vulnerabilities.

The entire process is structured as a multi-round game where the two teams drive the iterative adversarial adaptation of offensive and defensive capabilities. We established a closed-loop adversarial workflow to simulate the continuous evolution of system security, as illustrated in Figure 3. A detailed breakdown of the agent architectures, model selections, and the step-by-step procedure is provided in Appendix A.

The optimization process terminates if any of the following **stopping criteria** are met: (i) the failure of both the Red and Blue teams to generate any new structured findings (**null production**); (ii) an **execution failure** within the batch remediation module; (iii) **metric convergence**, defined as the vulnerability count remaining unchanged for three consecutive epochs (`Count_delay` $= 3$); or (iv) reaching the maximum number of **predefined epochs** (`max_epoch` $= 5$). To facilitate quantitative evaluation, we archive all interaction artifacts from each epoch and trace the vulnerability reduction trajectory, denoted as (C_{i-1}, C_i).

Metrics To rigorously validate the effectiveness of the RvB framework, we established a metric suite centered on defensive robustness. We measure the per-round **Defense Success Rate (DSR)** to evaluate the Blue Team's remediation capabilities and employ the **Attack Success Count (ASC)** as a quantitative proxy for Attack Complexity (AC). Crucially, to guarantee fidelity in automated remediation, we proposed a granular decomposition of DSR into **True DSR (TDSR)** and **Fake DSR (FDSR)**. This distinction allows us to isolate effective semantic fixes from destructive patches that technically mitigate vulnerabilities only by inducing service outages or interface failures. The formal definitions and calculation logic for these metrics are detailed in Appendix A.3.

Results and Analysis Based on the experiments described above, we obtained the following key results:

1) The RvB framework significantly enhances cybersecurity operational capabilities. As illustrated in Figure 4, the adversarial interaction effectively drives performance gains for both agents. We established a *Baseline* simulating a standard cooperative Multi-Agent System (MAS), where the Defense Success Rate (DSR) and Attack Success Count (ASC) are calculated as the cumulative union of results over the maximum epochs. Observations indicate that while the RvB framework's initial DSR was lower than the baseline, it rapidly improved, surpassing the baseline at Iteration 3 (80%) and reaching 90% by Iteration 5. Concurrently, the Red Team demonstrated improved At-

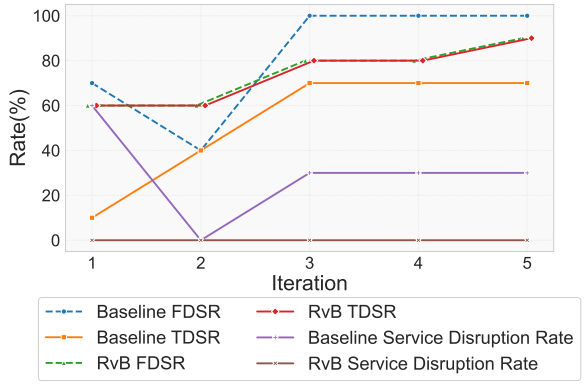

Figure 5: Comparative analysis of defense robustness between the Baseline and RvB frameworks. The solid lines track the True Defense Success Rate (TDSR), representing valid fixes, while the dashed lines indicate the Fake Defense Success Rate (FDSR). The purple line highlights the significant Service Disruption Rate (SDR) in the Baseline, caused by destructive patches. In contrast, the RvB framework (brown line) maintains a near-zero SDR, demonstrating that adversarial feedback ensures high-quality remediation without compromising service availability.

tack Capability (AC) through state updates, identifying new vulnerabilities and increasing the ASC from 5 to 7. These results validate that the RvB paradigm effectively elevates the capabilities of both offensive and defensive agents compared to a static cooperative baseline.

2) The RvB framework improves the safety and robustness of the multi-agent system. As shown in Figure 5, the system employing the RvB framework exhibits a substantially lower service drop rate. To quantify this, we defined three metrics: *True DSR (TDSR)*, representing cases where the vulnerability is fixed and the service remains functional; *Fake DSR (FDSR)*, where the attack check fails potentially due to service destruction (e.g., the Blue Team deleting critical files); and *Service Disruption Rate (SDR)*, calculated as the difference between FDSR and TDSR. Remarkably, the RvB framework maintains an SDR of 0 throughout the process. In contrast, the cooperative baseline exhibits an SDR as high as 60%. Analysis of execution logs reveals that the Red Team's directed adversarial reports provide precise, actionable contexts, preventing the Blue Team from making indiscriminate or destructive modifications to the codebase. This confirms that the adversarial feedback loop is crucial for ensuring valid, non-disruptive remediation.

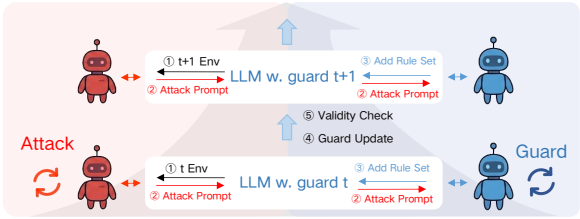

Figure 6: Overview of the iterative adversarial dialogue for guardrail optimization. The interaction cycle proceeds as follows: (1) The Red Team probes the current environment state; (2) it executes an adaptive attack prompt aimed at bypassing the guard; (3) The Blue Team analyzes the successful exploit to synthesize a targeted rule set; (4) The guardrail undergoes a dynamic update, transitioning the system to a more robust state $t + 1$; and (5) A validity check using benign queries is performed to prevent over-interception and ensure the preservation of model utility.

4.2 The Jailbreaker vs. Guardrail in Content Security

Experiment Setting To evaluate the RvB framework in a realistic security context, we designed an experiment simulating an adversarial dialogue between a *Jailbreaker* (Red Team) and a *Guardrail* (Blue Team). We selected harmful prompts from the HarmBench benchmark (Mazeika et al., 2024) as the attack targets. The Red Team agent, built upon the **Composition-of-Principles (CoP) framework** (Xiong et al., 2025a), iteratively refines attack strategies to generate effective jailbreak prompts. The Blue Team agent is an adaptive guardrail system based on **NeMo Guardrails** (Rebedea et al., 2023), which can dynamically augment its rule set to defend against novel attacks.

The entire process is structured as a multi-round game where the two teams drive the iterative adversarial adaptation of offensive and defensive capabilities, as illustrated in Figure 6. A detailed breakdown of the agent architectures, model selections, and the step-by-step procedure is provided in Appendix B.

Metrics To quantitatively measure the dynamics of this adversarial interplay, we designed a set of corresponding metrics. To maintain a consistent defensive perspective, our primary metrics focus on the Blue Team's performance. We measure the **Defense Success Rate (DSR)** against the Red Team's latest attacks in each round, and the **Average Attack Turns (AAT)** the Red Team requires for a successful breach. For a more granular analysis of learning capabilities, we introduce the

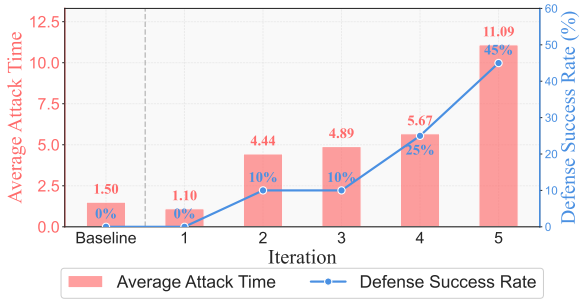

Figure 7: Defense Success Rate and Average Attack Time comparison. As iterations progress, DSR (bars, right y-axis) consistently rises, while AAT (line, left y-axis) also trends upward, indicating that the Blue Team's guardrail is becoming progressively stronger and more difficult to bypass.

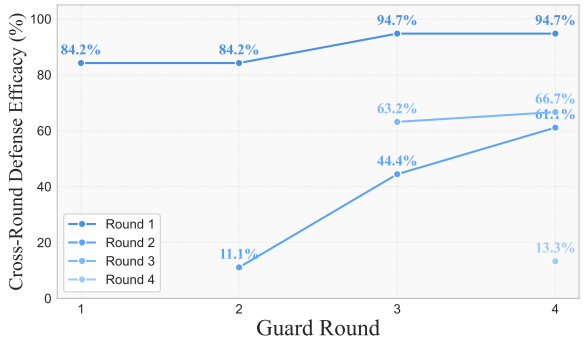

Figure 8: Cross-Round Defense Efficacy (CRDE) for exp. 2. Each line shows that newer guardrails (e.g., Round 4) perform better against attacks from all rounds, including older ones (e.g., Round 1), indicating no catastrophic forgetting.

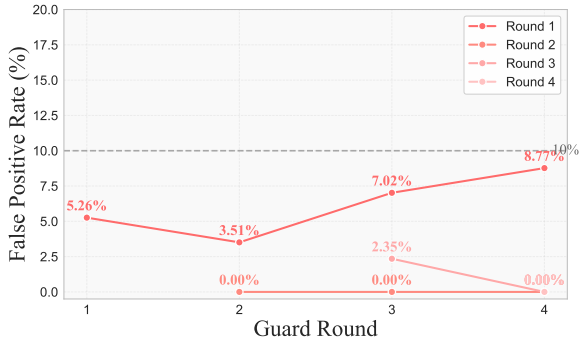

Figure 9: False Positive Rate (FPR) for exp. 2. The FPR remains consistently low across all rounds, indicating minimal over-interception of benign prompts.

Cross-Round Defense Efficacy (CRDE) to test for catastrophic forgetting, and the **False Positive Rate (FPR)** to assess over-interception. The formal definitions of these metrics are provided in Appendix C.

Results and Analysis By compiling the metrics across four adversarial rounds, we obtained the following key results:

1) The RvB framework drives continuous and significant security improvement. As depicted in Figure 7, the Blue Team's Defense Success Rate (DSR) against the Red Team's attacks shows a clear upward trend with each adversarial round, climbing from 0% in the initial rounds to 45% by the final iteration. This indicates that the guardrail's ability to intercept novel attacks is continuously improving. Concurrently, the Average Attack Turns (AAT) required for a successful breach also steadily rise, ultimately reaching 11.09 turns in the final round—a more than 7-fold increase from the baseline of 1.50. This further confirms that the Red Team's task is becoming progressively more difficult. The task-level analysis in Figure 11 (in Appendix) corroborates this, showing that the number of attempts required for successful attacks increases substantially in later rounds for most tasks.

2) The Blue Team learns generalizable rules without catastrophic forgetting. A crucial aspect of iterative learning is avoiding overfitting to the most recent attacks. Figure 8 illustrates the Cross-Round Defense Efficacy (CRDE), showing how guardrails from later rounds perform against attacks from earlier rounds. Each line represents the performance of a progressively updated guardrail (from Guard Round 1 to 4) tested against a fixed set of historical attacks (from Round 1, 2, 3, or 4). A clear upward trend is observable across all lines, demonstrating two key findings. Firstly, the guardrail's ability to defend against new attacks improves with each iteration. Secondly, and more importantly, its defense performance against older attack sets (e.g., the Guard Round 4's performance on Round 1 attacks) does not degrade but consistently improves. This indicates our iterative augmentation mechanism successfully avoids catastrophic forgetting. Furthermore, as shown in Figure 9, the False Positive Rate (FPR) remains at a very low level, demonstrating that the enhanced security does not come at the cost of excessive over-interception.

3) The evolved guardrail demonstrates strong out-of-domain generalization. To verify that the learned defensive capabilities generalize beyond the specific attack patterns seen during interaction, we evaluated the initial and final guardrails on 100 randomly selected prompts from four external, unseen jailbreak benchmarks. As shown in Figure 10,

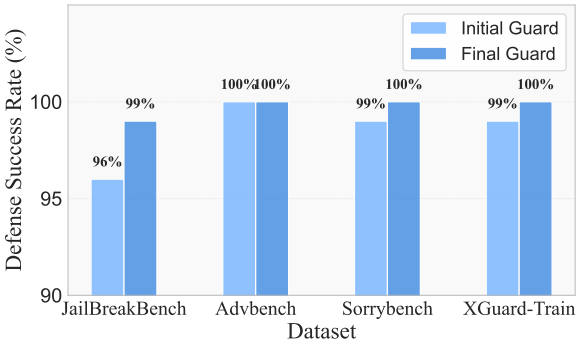

Figure 10: Out-of-Domain performance comparison between the initial and final guardrail. The final guardrail shows significantly improved defensive performance on four unseen external benchmarks.

the Final Guard consistently outperforms the Initial Guard across all external datasets (JailBreakBench (Chao et al., 2024), AdvBench (Zou et al., 2023), SorryBench (Xie et al., 2024), and XGuard-Train (Upadhayay et al., 2025)), with its defense success rate improving from 96% to 99% on JailBreakBench and achieving a perfect 100% on AdvBench. This indicates that the adversarial iterations compelled the Blue Team to learn more fundamental and robust principles of defense, enhancing its resilience against a broader spectrum of unseen jailbreak attacks.

5 Discussion

Our findings substantiate the efficacy of the **RvB** framework, offering improvements that stem directly from the game-theoretic structure rather than mere model scaling. We analyze the mechanisms driving these results below.

Convergence to Practical Equilibrium. The adversarial interaction drives the system toward a *practical equilibrium* where defensive coverage matches the high-probability attack manifold. As the Red Team's belief entropy ($H(b_k)$) decreases, its strategy shifts from broad exploration to focused exploitation. The resulting convergence of high Defense Success Rates with escalating Attack Costs suggests that further exploitation becomes computationally prohibitive. While a formal Nash Equilibrium in the high-dimensional discrete space of natural language remains theoretically elusive, the monotonic state progression confirms that offensive and defensive utilities balance at a robust security standard.

Interaction-Driven Generalization. The strong performance on out-of-domain attacks (Figure 10) indicates that robustness emerges from the adversarial dynamic rather than latent model knowledge alone. In this zero-sum loop, the Red Team actively probes for blind spots, functioning as a continuous regularizer. This pressure penalizes superficial pattern matching, forcing the Blue Team to induce abstract, invariant security principles—such as detecting malicious intent structures over specific keywords. The interaction effectively synthesizes a curriculum of "hard negatives" from the tail of the vulnerability distribution, driving generalization without explicit supervision.

Operational Viability via Semantic Verification. Crucially, the adversarial pairing resolves the "destructive remediation" bottleneck often found in autonomous software engineering. The near-zero Service Disruption Rate (SDR) confirms that the Red Team serves as a stringent semantic verifier. By coupling remediation with immediate, targeted exploitation, the framework imposes a check-and-balance dynamic absent in cooperative systems, ensuring that security hardening does not compromise system availability.

6 Conclusion

In this work, we present a unified **Red Team vs. Blue Team (RvB)** framework that automates the continuous hardening of AI systems. By formalizing security operations as a sequential, imperfect-information game, we bridge the gap between offensive discovery and defensive mitigation, enabling a self-reinforcing loop of capability enhancement without model tuning.

Experiments on cyber and content security demonstrate that this paradigm shifts security automation from static, post-hoc patching to iterative adversarial adaptation. The framework not only successfully repairs vulnerabilities with high precision and minimal service disruption, but also produces defensive guardrails with strong out-of-distribution generalization. These findings suggest that the most effective way to secure foundation models is to subject them to the adversarial pressures they face in real-world scenarios, governed by a structured game-theoretic architecture.

Future work will explore the theoretical bounds of this convergence in multi-modal environments and investigate the application of this framework to more complex, long-horizon security tasks.

Limitations

The efficacy of the RvB framework relies intrinsically on the reasoning capabilities of the underlying foundation models. Regarding evaluation scope, our current experiments utilized a representative but constrained set of web vulnerabilities. Rigorous testing against larger-scale, diverse benchmarks is required to further verify generalizability. We also note that the turn-based simulation serves as a controlled abstraction, effectively simplifying the asynchronous dynamics inherent in real-world security operations.

Ethics Statement

This work involves the development of automated agents capable of executing software exploits and generating adversarial jailbreaks. We acknowledge the dual-use nature of such technologies; while intended for system hardening, the offensive capabilities of the Red Team agent could theoretically be repurposed for malicious attacks. To mitigate these risks, we strictly adhered to the following safety protocols:

Controlled Environments. All cybersecurity experiments were conducted within isolated, network-gated Docker containers with no internet access beyond the local simulation loop. This ensures that the automated exploit generation cannot propagate to external systems or critical infrastructure.

Defensive Focus. The primary objective of the RvB framework is to minimize the epistemic uncertainty of the defender (Blue Team). The offensive agent functions solely as a diagnostic tool to drive this defensive adaptation. We do not release offensive agent that function independently of the hardening loop.

Responsible Disclosure and Data. For the content security tasks, we utilized public benchmarks (e.g., HarmBench) and strictly monitored the generated outputs. No new private datasets of harmful content were distributed. Our code release includes safeguards to prevent the standalone deployment of the Red Team agent for malicious probing. We believe that documenting these automated red-teaming dynamics is essential for the community to develop robust defenses against emerging autonomous threats.

References

Nirmit Arora, Sathvik Joel, Ishan Kavathekar, Rohan Gandhi, Yash Pandya, Tanuja Ganu, Aditya Kanade, Akshay Nambi, and 1 others. 2025. Exposing weak links in multi-agent systems under adversarial prompting. *arXiv preprint arXiv:2511.10949*.

Sebastián R. Castro, Roberto Campbell, Nancy Lau, Octavio Villalobos, Jiaqi Duan, and Alvaro A. Cardenas. 2025. Large language models are autonomous cyber defenders. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, page 1125–1132. IEEE.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramer, Hamed Hassani, and Eric Wong. 2024. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *Preprint*, arXiv:2404.01318.

David de Fitero-Dominguez, Eva Garcia-Lopez, Antonio Garcia-Cabot, and Jose-Javier Martinez-Herraiz. 2024. Enhanced automated code vulnerability repair using large language models. *Engineering Applications of Artificial Intelligence*, 138:109291.

Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2024. Pentestgpt: An llm-empowered automatic penetration testing tool. *Preprint*, arXiv:2308.06782.

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. *Preprint*, arXiv:2212.14834.

Ali Dorri, Salil S Kanhere, and Raja Jurdak. 2018. Multi-agent systems: A survey. *Ieee Access*, 6:28573–28593.

Falong Fan and Xi Li. 2025. Peerguard: Defending multi-agent systems against backdoor attacks through mutual reasoning. *arXiv preprint arXiv:2505.11642*.

Pengfei He, Yuping Lin, Shen Dong, Han Xu, Yue Xing, and Hui Liu. 2025. Red-teaming llm multi-agent systems via communication attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6726–6747.

Julius Henke. 2025. Autopentest: Enhancing vulnerability management with autonomous llm agents. *Preprint*, arXiv:2505.10321.

Monte Hoover, Vatsal Baherwani, Neel Jain, Khalid Saifullah, Joseph Vincent, Chirag Jain, Melissa Kazemi Rad, C. Bayan Bruss, Ashwinee Panda, and Tom Goldstein. 2025. Dynaguard: A dynamic guardian model with user-defined policies. *Preprint*, arXiv:2509.02563.

Trung-Kiet Huynh, Duy-Minh Dao-Sy, Thanh-Bang Cao, Phong-Hao Le, Hong-Dan Nguyen, Phu-Quy Nguyen-Lam, Minh-Luan Nguyen-Vo, Hong-Phat Pham, Phu-Hoa Pham, Thien-Kim Than, and 1 others. 2025. Understanding llm agent behaviours via game theory: Strategy recognition, biases and multi-agent dynamics. *arXiv preprint arXiv:2512.07462*.

Yoichi Ishibashi and Yoshimasa Nishimura. 2024. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*.

Kunal Jha, Wilka Carvalho, Yancheng Liang, Simon S Du, Max Kleiman-Weiner, and Natasha Jaques. 2025. Cross-environment cooperation enables zero-shot multi-agent coordination. *arXiv preprint arXiv:2504.12714*.

Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. Inferfix: End-to-end program repair with llms. *Preprint*, arXiv:2303.07263.

Donghyun Lee and Mo Tiwari. 2024. Prompt infection: Llm-to-llm prompt injection within multi-agent systems. *arXiv preprint arXiv:2410.07283*.

Miles Q Li and Benjamin CM Fung. 2025. Security concerns for large language models: A survey. *Journal of Information Security and Applications*, 95:104284.

Xu Liu, Yan Chen, Kan Ling, Yichi Zhu, Hengrun Zhang, Guisheng Fan, and Huiqun Yu. 2025. An automated framework for strategy discovery, retrieval, and evolution in llm jailbreak attacks. *arXiv preprint arXiv:2511.02356*.

Víctor Mayoral-Vilches, Luis Javier Navarrete-Lozano, María Sanz-Gómez, Lidia Salas Espejo, Martiño Crespo-Álvarez, Francisco Oca-Gonzalez, Francesco Balassone, Alfonso Glera-Picón, Unai Ayucar-Carbajo, Jon Ander Ruiz-Alcalde, Stefan Rass, Martin Pinzger, and Endika Gil-Uriarte. 2025. Cai: An open, bug bounty-ready cybersecurity ai. *Preprint*, arXiv:2504.06017.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, and 1 others. 2024. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*.

Salman Rahman, Liwei Jiang, James Shiffer, Genglin Liu, Sheriff Issaka, Md Rizwan Parvez, Hamid Palangi, Kai-Wei Chang, Yejin Choi, and Saadia Gabriel. 2025. X-teaming: Multi-turn jailbreaks and defenses with adaptive multi-agents. *arXiv preprint arXiv:2504.13203*.

Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. In *Proceedings of the 2023 conference on empirical methods in natural language processing: system demonstrations*, pages 431–445.

Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. 2025. Pentestagent: Incorporating llm agents to automated penetration testing. *Preprint*, arXiv:2411.05185.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. Magis: Llm-based multi-agent framework for github issue resolution. *Advances in Neural Information Processing Systems*, 37:51963–51993.

Bibek Upadhayay, Vahid Behzadan, and Ph. D. 2025. X-guard: Multilingual guard agent for content moderation. *Preprint*, arXiv:2504.08848.

Zhefan Wang, Yuanqing Yu, Wendi Zheng, Weizhi Ma, and Min Zhang. 2024. Multi-agent collaboration framework for recommender systems. *arXiv preprint arXiv:2402.15235*.

Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, page 1–13. ACM.

Tinghao Xie, Xiangyu Qi, Yi Zeng, Yangsibo Huang, Udari Madhushani Sehwag, Kaixuan Huang, Luxi He, Boyi Wei, Dacheng Li, Ying Sheng, and 1 others. 2024. Sorry-bench: Systematically evaluating large language model safety refusal. *arXiv preprint arXiv:2406.14598*.

Chen Xiong, Pin-Yu Chen, and Tsung-Yi Ho. 2025a. Cop: Agentic red-teaming for large language models using composition of principles. *arXiv preprint arXiv:2506.00781*.

Chen Xiong, Xiangyu Qi, Pin-Yu Chen, and Tsung-Yi Ho. 2025b. Defensive prompt patch: A robust and interpretable defense of llms against jailbreak attacks. *Preprint*, arXiv:2405.20099.

Guangyu Yang, Jinghong Chen, Jingbiao Mei, Weizhe Lin, and Bill Byrne. 2025a. Retrieval-augmented defense: Adaptive and controllable jailbreak prevention for large language models. *Preprint*, arXiv:2508.16406.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024a. Swe-agent: Agent-computer interfaces enable automated software engineering. *Preprint*, arXiv:2405.15793.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024b. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Xianglin Yang, Gelei Deng, Jieming Shi, Tianwei Zhang, and Jin Song Dong. 2025b. Enhancing model defense against jailbreaks with proactive safety reasoning. *Preprint*, arXiv:2501.19180.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. 2025c. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *arXiv preprint arXiv:2504.00587*.

Zonghao Ying, Deyue Zhang, Zonglei Jing, Yisong Xiao, Quanchen Zou, Aishan Liu, Siyuan Liang, Xiangzheng Zhang, Xianglong Liu, and Dacheng Tao. 2025. Reasoning-augmented conversation for multi-turn jailbreak attacks on large language models. *arXiv preprint arXiv:2502.11054*.

Yifan Zeng, Yiran Wu, Xiao Zhang, Huazheng Wang, and Qingyun Wu. 2024. Autodefense: Multi-agent llm defense against jailbreak attacks. *arXiv preprint arXiv:2403.04783*.

Andy Zhou, Bo Li, and Haohan Wang. 2024. Robust prompt optimization for defending language models against jailbreaking attacks. *Preprint*, arXiv:2401.17263.

Junda Zhu, Lingyong Yan, Shuaiqiang Wang, Dawei Yin, and Lei Sha. 2025. Reasoning-to-defend: Safety-aware reasoning can defend large language models from jailbreaking. *Preprint*, arXiv:2502.12970.

Andy Zou, Maxwell Lin, Eliot Jones, Micha Nowak, Mateusz Dziemian, Nick Winter, Alexander Grattan, Valent Nathanael, Ayla Croft, Xander Davies, and 1 others. 2025. Security challenges in ai agent deployment: Insights from a large scale public competition. *arXiv preprint arXiv:2507.20526*.

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

A Experiment Details of The Attacker vs. Defender in Cyber Security

A.1 Red Team Setting

The architecture of the red team agent (CAI) consists of three core components: a *Planner* responsible for deducing attack paths, an *Executor* for executing specific Bash/MCP commands, and a *Reporter* for summarizing attack outcomes. During the offensive process, the red team first performs passive reconnaissance on the target environment. The *Planner* generates attack hypotheses based on system feedback, after which the *Executor* invokes tools to conduct active probing and payload delivery. Upon a successful exploit, the agent maintains access and triggers the *Reporter* to generate a vulnerability report containing reproduction steps. For this experiment, we selected `gemini-3.0-flash` as the backbone model. The maximum number of interaction turns was set to 30 to ensure sufficient probing depth.

A.2 Blue Team Setting

The blue team's Mini-SWE-Agent is designed to simulate the remediation workflow of a security engineer. Its inputs are the vulnerability report generated by the red team and the project's source code. The blue team's workflow comprises three stages: *fault localization*, *patch generation*, and *regression verification*. First, the agent analyzes the codebase based on the vulnerability report to locate the vulnerable PHP files. Next, it generates a patch in `git diff` format and applies it to the environment. Finally, it restarts the Docker container to verify service availability and confirm the mitigation of the vulnerability. To ensure the accuracy of code modifications and prevent disruption of existing business logic, the blue team model requires a high level of code comprehension and generation capability; thus, we also selected `gemini-3.0-flash` as its underlying model. If the initial repair fails, the blue team is permitted a maximum of 3 retry attempts to correct the patch.

A.3 Metric Definitions for Cyber Security

In the context of the Cyber Security (Code Hardening) task, let $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$ denote the set of vulnerability test cases. For each test case x_i at a given interaction round, we define the outcome state as a tuple $(r_{att}^{(i)}, r_{reg}^{(i)})$, where:

- $r_{att}^{(i)} \in \{0, 1\}$ denotes the outcome of the Red Team's attack. $r_{att}^{(i)} = 1$ indicates a successful breach (exploit succeeded), while 0 indicates a failed attack (defense succeeded).
- $r_{reg}^{(i)} \in \{0, 1\}$ denotes the outcome of the service regression test. $r_{reg}^{(i)} = 1$ indicates the service remains functional, while 0 indicates a service disruption (e.g., HTTP 500 error or missing interface).

Defensive Metrics: DSR, TDSR, and FDSR The **Defense Success Rate (DSR)** represents the overall ratio of cases where the Blue Team successfully fixes the system.
True Defense Success Rate (TDSR): The proportion of cases where the vulnerability is mitigated

and the service integrity is preserved.

$$\text{DSR} = \text{TDSR} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(r_{att}^{(i)} = 0 \wedge r_{reg}^{(i)} = 1) \tag{5}$$

Fake Defense Success Rate (FDSR): The proportion of cases where the attack fails solely because the defensive patch rendered the service non-functional.

$$\text{FDSR} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(r_{att}^{(i)} = 0) \tag{6}$$

Service Disruption Rate (SDR): A measure of availability loss caused by defensive over-optimization. It denotes the frequency with which valid user requests are blocked or service interfaces become unresponsive as a direct result of the applied security patches.

$$\text{SDR} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(r_{att}^{(i)} = 0 \wedge r_{reg}^{(i)} = 0) \tag{7}$$

From the equations above, it holds that SDR = FDSR − TDSR. A robust automated defense agent should maximize TDSR while maintaining SDR near zero.

Offensive Metric: Attack Success Count (ASC) We utilize **Attack Success Count (ASC)** to measure the diversity and complexity of the attack vectors discovered by the Red Team. Let $\mathcal{E}_k$ be the set of unique, valid exploit payloads successfully executed by the Red Team up to round k. The ASC is defined as the cardinality of this cumulative set:

$$\begin{aligned}\text{ASC}_k &= |\mathcal{E}_k| \\ &= \left|\bigcup_{t=1}^{k}\{e \mid e \in \text{Payloads}_t, \text{IsSuccess}(e)\}\right|\end{aligned} \tag{8}$$

Higher ASC values indicate that the Red Team is capable of evolving its strategy to uncover deeper logic vulnerabilities as the environment hardens.

A.4 Detailed Procedure

1. **Attack Generation (Red Team):** The Red agent initiates the cycle by interacting with the current environment (t Env). It performs reconnaissance and exploitation to identify security flaws. Upon successful compromise, it produces a structured *Attack Log* containing the vulnerable file path, the snippet of problematic code, and the root cause of the bug. A representative log entry is formatted as follows:

```
{
"file": "php_action/removeOrder.php",
"code":       "$orderId=    $_GET['id'];\n
if($orderId) { \n $sql = \"UPDATE orders
SET order_status = 2 WHERE order_id =
{$orderId}\"; ... }",
"bug": "SQL Injection via 'id' parameter
in GET request. The variable $orderId is
interpolated without sanitization.",
"payload": "id=1 OR 1=1"
}
```

   This structured output details the exploit methodology and reproduction steps, serving as the ground truth for the subsequent fix.

2. **Remediation (Blue Team):** Acting on the information provided in the *Attack Log*, the Blue agent analyzes the codebase to localize the fault. It then synthesizes a code patch and performs a *Fix Act* to repair the vulnerability.

3. **State Update & Verification:** The generated patch is applied to the Docker container, transitioning the system to *Target State* $t+1$. Finally, a *Vuln Check* is executed to verify the effectiveness of the fix and ensure no regression errors were introduced. If the vulnerability persists, the Blue agent is triggered to refine its patch in subsequent iterations.

B Experiment Details of The Jailbreaker vs. Guardrail in Content Security

B.1 Agent Architecture and Model Selection

The Red and Blue teams were designed with specialized components and models to fulfill their respective roles in the adversarial framework.

Red Team Components The Red Team agent, based on the Composition-of-Principles (CoP) framework, includes four main LLM-driven components:

- **Target Model (*target_model*):** The model to be attacked, protected by the Blue Team's guardrail. To better observe the iterative improvement of the guardrail, this model was selected to have relatively weak intrinsic safety features but adequate question-answering capabilities. We chose **Qwen2-7B-Instruct**.

- **Helper Model (*helper_model*):** Responsible for generating diverse attack strategies. This role requires high capability without being prone to refusal. We chose **DeepSeek-V3**.
- **Attack Model (*attack_model*):** Guided by the strategies from the helper model, this component generates the actual attack prompts.
- **Judge Model (*judge_model*):** Evaluates the success of an attack by checking if the target model's response aligns with the harmful intent, while also monitoring for topic drift.

For the *attack_model* and *judge_model*, we required highly capable open-source models that could reliably follow complex instructions. We chose **Qwen3-32B** for both roles.

Blue Team Components The Blue Team agent, based on NeMo Guardrails, consists of the following:

- **Target Model (*target_model*):** The same **Qwen2-7B-Instruct** used by the Red Team.
- **Guard Model (*guard_model*):** Used during the defense phase to classify incoming prompts based on the current rule set.
- **Patch Model (*patch_model*):** Used to analyze successful attack prompts and augment the rule set with more general, robust rules.

Both the *guard_model* and *patch_model* require stable, high-quality reasoning and are less likely to trigger refusal mechanisms. Therefore, we leveraged a powerful closed-source model with strong safety alignments, selecting **GPT-4o-mini** for these roles.

Validation Component To prevent the augmented rules from overfitting, a validation step is performed using negative samples. The *generate_model* for creating these benign samples needs to be capable yet avoid refusal. We chose **DeepSeek-V3** for this role. For each positive (harmful) prompt, three corresponding negative (benign) samples were generated.

B.2 Detailed Procedure

The adversarial interaction proceeds as follows:

1. The Red Team is given a set of harmful prompts (e.g., from HarmBench). For each prompt, it conducts up to 3 outer rounds of strategy refinement, with up to 10 inner attack attempts per round, to generate a set of effective jailbreak prompts that bypass the Blue Team's current guardrail ($Guard_{i-1}$).
2. The Blue Team receives the set of successful jailbreak prompts ($\mathcal{A}_i$) from the Red Team. It uses the *patch_model* to analyze these prompts, identify common patterns, and generate a new, more generalized set of defensive rules.
3. The Blue Team integrates these new rules into its existing rule set to form an updated guardrail, $Guard_i$.
4. To prevent overfitting and ensure the new rules do not incorrectly block harmless queries, the Blue Team validates $Guard_i$ against a set of benign prompts ($\mathcal{D}_{benign,i}$).
5. The process repeats, with the Red Team now facing the hardened guardrail $Guard_i$ in the next iteration ($i + 1$). A detailed flowchart of this procedure is shown in Figure X.

C Metric Definitions

Here we provide the formal definitions for the metrics used in our evaluation. Let $\mathcal{D}_{harm}$ be the set of harmful test samples ($N_{total} = |\mathcal{D}_{harm}|$). Let $\mathcal{A}_j$ be the set of effective attack prompts generated by the red team in iteration j. Let $\mathcal{D}_{benign,j}$ be the set of benign (negative) samples for validation in iteration j. Let $Guard_i$ be the blue team's guardrail system after iteration i.

- **Defense Success Rate (DSR):** Our primary metric, measuring the proportion of harmful samples in round i that are successfully intercepted by the guardrail from the previous state, $Guard_{i-1}$. It is the inverse of the Attack Success Rate (ASR), calculated as $1 - \text{ASR}_i$.

$$\text{DSR}_i = 1 - \frac{1}{N_{total}} \sum_{q \in \mathcal{D}_{harm}} \text{IsSuccess}(p_{q,i}, G_{i-1}) \tag{9}$$

  where $\text{IsSuccess}(p, G)$ returns 1 if prompt p bypasses guardrail G.

- **Average Attack Turns (AAT):** The average number of inner attack attempts the Red Team requires for a successful jailbreak in round i.

$$\text{AAT}_i = \frac{\sum_{q \in \mathcal{S}_i} k_{q,i}}{|\mathcal{S}_i|} \tag{10}$$

where $\mathcal{S}_i$ is the subset of successfully attacked samples and $k_{q,i}$ is the number of attempts for sample q.

- **Cross-Round Defense Efficacy (CRDE):** Formerly DSR, this metric measures the capability of the guardrail at iteration i to defend against the set of effective attacks from a previous or current iteration j (where $j \leq i$). It tests for catastrophic forgetting.

$$\text{CRDE}_{i,j} = \frac{1}{|\mathcal{A}_j|} \sum_{p \in \mathcal{A}_j} (1 - \text{IsSuccess}(p, G_i)) \quad (11)$$

- **False Positive Rate (FPR):** The proportion of harmless samples from $\mathcal{D}_{benign,j}$ that are incorrectly blocked by $Guard_i$.

$$\text{FPR}_{i,j} = \frac{\sum_{p_b \in \mathcal{D}_{benign,j}} \text{IsBlocked}(p_b, \text{Guard}_i)}{|\mathcal{D}_{benign,j}|} \quad (12)$$

where $\text{IsBlocked}(p_b, G)$ returns 1 if the benign prompt p_b is incorrectly intercepted.

D Additional Results and Figures

This section provides supplementary figures referenced in the main body. Figure 11 provides a task-level breakdown of the attack difficulty across the adversarial rounds.

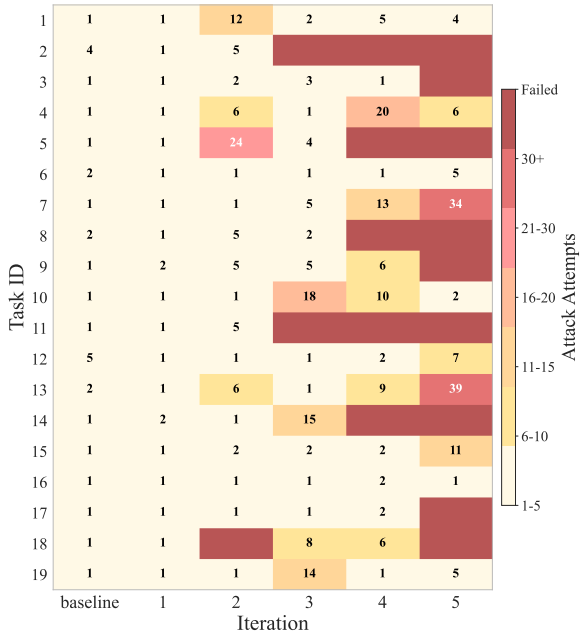

Figure 11: The Attack Time (number of attempts) of each task across all the adversarial rounds. A clear trend of increasing attack difficulty (color shifting from yellow to red/darker) is visible for most tasks in later iterations, complementing the aggregate AAT trend shown in Figure 7.

E Cost Analysis

To evaluate the operational viability of the RvB framework, we conduct a comprehensive analysis of computational costs in terms of token consumption. Contrary to the intuition that iterative adversarial loops increase overhead, our results demonstrate that the RvB paradigm is more resource-efficient than traditional cooperative Multi-Agent Systems (MAS).

E.1 The Attacker vs. Defender in Cyber Security

Table 1 details the token usage across the same experimental settings(Max_epoch=5). The RvB framework consumes a total of 5,105,759 tokens, representing an **18.35%** reduction in total computational cost compared to the Cooperative MAS (Pure Blue) baseline (6,253,192 tokens).

We attribute this efficiency to the *precision* of adversarial feedback. In a cooperative MAS environment, the defender often performs broad, indiscriminate reasoning to identify vulnerabilities, leading to high input token counts. Conversely, the Red Team in the RvB framework provides targeted, structured attack logs that serve as precise actionable contexts. This focused evidence prunes the Blue Team's search space, which is mathematically equivalent to reducing the epistemic uncertainty and entropy of the defense policy. Consequently, the Blue Team achieves higher remediation quality with fewer redundant reasoning cycles.

Table 1: Comparison of cumulative token consumption during the hardening process. The RvB framework significantly reduces input token overhead by leveraging targeted adversarial reports, which prevent redundant exploratory reasoning in the Blue Team.

Setting	Input	Output	Total
Pure Red (Offense)	576,140	17,742	593,882
Pure Blue (Cooperative MAS)	5,943,099	310,093	6,253,192
RvB (Ours)	**4,800,430**	**305,329**	**5,105,759**

E.2 The Jailbreaker vs. Guardrail in Content Security

In the content security experiment, we analyze the computational cost to demonstrate the framework's

efficiency and accessibility. A key characteristic of this setup is its primary reliance on high-capability open-source models, allowing for local deployment. To provide a tangible measure of the resources consumed, we estimate the hypothetical cost of each round using publicly available reference API prices[2]. The pricing for all models utilized in this experiment is detailed in Table 2.

Table 2: Reference API pricing for models used in the content security experiment. Prices are listed per million (M) tokens.

Model	Input Prices	Output Prices
DeepSeek-V3	$0.2820	$1.1270
Qwen3-32B	$0.1410	$0.5640
GPT-4o-mini	$0.0970	$0.3870
Qwen2-7B-Instruct	$0.0000	$0.0000

Table 3 details the total token usage and the corresponding estimated costs for a static baseline and each iterative round of the RvB process. The costs are calculated by applying the reference prices from Table 2 to the precise input and output token consumption of each model involved in each stage.

Table 3: Comparison of token consumption and estimated costs per round for the content security experiment.

Round / Setting	Total Tokens	Estimated Cost ($)
Baseline	4,040,889	$1.42
RvB Round 1	2,828,789	$1.00
RvB Round 2	5,961,159	$2.10
RvB Round 3	10,490,319	$3.70
RvB Round 4	12,223,668	$4.31

The results clearly indicate the economic viability of our framework. Even the most computationally intensive iteration, Round 4, has a hypothetical cost of only $4.31. This demonstrates that, even when benchmarked against commercial API rates, the per-round overhead of our iterative adversarial framework is well within an acceptable and modest range for academic research and practical application. This highlights the framework's accessibility, making advanced, co-evolutionary security hardening a financially feasible endeavor.

[2]Reference pricing data sourced from Third-party Platform