RvB: Training-Free Artifact Hardening through Iterative Red–Blue Feedback

Anonymous authors
Paper under double-blind review

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.

1 Introduction

The rapid proliferation of large language models (LLMs) has introduced a fundamental duality to security (Li & Fung, 2025). On one side, autonomous agents such as Pentest-Agent (Shen et al., 2025) and CAI (Mayoral-Vilches et al., 2025) can execute multi-step offensive workflows, while LLM-based methods can synthesize adaptive jailbreaks (Xiong et al., 2025a; Rahman et al., 2025). On the other side, coding agents such as SWE-agent (Yang et al., 2024), autonomous incident-response systems (Castro et al., 2025), and dynamic guardrails (Zeng et al., 2024) use similar model capabilities for remediation and harmful-content prevention.

Despite these parallel advances, offensive and defensive systems are often studied in separate evaluation settings. Defenses are often tested against static attack sets or through post-hoc analysis (Chao et al., 2024), whereas offensive agents usually probe a target that does not respond with a persistent repair. Once an artifact changes, however, the old attack set may become stale and the update may expose different weaknesses. This motivates a dynamic protocol in which attack discovery and remediation repeatedly condition one another. From an AI perspective, the resulting problem is environment-mediated adaptation: the agents remain fixed, but their subsequent behavior is elicited against a target altered by earlier interactions.

We therefore propose the **Red Team vs. Blue Team (RvB)** framework, a training-free sequential interaction over a persistent artifact. In each round, the Red Team probes the current environment and returns attacks accepted by a task-specific checker or judge. The Blue Team uses this evidence together with the current system state to propose a code or rule update. The resulting artifact becomes the environment for the next round, so the next Red invocation tests the consequences of the preceding defense rather than only a frozen benchmark.

We evaluate this framework in two domains:

- **Code hardening:** CAI probes two vulnerable PHP services, and Mini-SWE-Agent uses accepted attack records to localize faults, apply source-level patches, restart the service, and run post-update vulnerability and regression checks.

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.

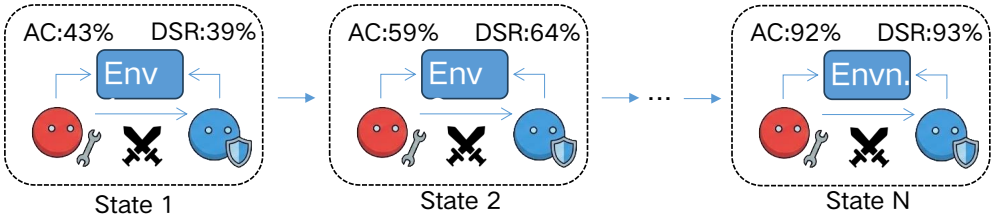


Figure 1: Schematic RvB interaction across persistent artifact states. Fixed Red and Blue agents repeatedly probe and update the shared environment from state S_1 to S_N.

- **Guardrail optimization:** a Composition-of-Principles jailbreaker repeatedly attacks a NeMo-based guardrail whose external rule set is revised from successful bypasses and screened on benign inputs.

The recorded code trajectories reach 90% and 100% defense success on the two systems, compared with cooperative summaries of 70% and 60%; because their aggregation conventions differ, these values are descriptive rather than a matched superiority estimate. In guardrail optimization, current-round defense success rises from 0% to 45%, the mean number of attempts among successful attacks increases from 1.50 to 11.09, and measured false-positive rates remain between 0% and 8.77%. The code trajectories include an intermediate regression and a plateau, while retained-attack replay measures protection on fixed historical attack sets. Together, these cases show how persistent artifact state and renewed adversarial probing make progress, plateaus, and regressions measurable without updating model weights.

2 Related Work

LLM multi-agent interaction. LLM-based multi-agent systems commonly divide a task among cooperating roles or use debate and coordination to improve a shared answer (Tao et al., 2024; Ishibashi & Nishimura, 2024; Yang et al., 2025b). Security changes the interaction: an attacker searches for failures while a defender must preserve both safety and utility. Existing adversarial-agent work covers prompt injection, communication attacks, jailbreak defense, and backdoor detection (Lee & Tiwari, 2024; He et al., 2025; Arora et al., 2025; Zeng et al., 2024; Fan & Li, 2025). RvB is neither symmetric self-play nor a parameter-learning game: its output is an evaluated update to a persistent external artifact.

Vulnerability discovery and repair. PentestGPT and AutoPentest decompose offensive procedures, while TitanFuzz and Fuzz4All generate inputs that expose software failures (Deng et al., 2024; Henke, 2025; Deng et al., 2023; Xia et al., 2024). InferFix and autonomous coding agents localize faults and synthesize patches (Jin et al., 2023; Yang et al., 2024); comparative work also evaluates Code Llama and Mistral for vulnerability repair (de Fitero-Dominguez et al., 2024). VRpilot studies reasoning and patch-validation feedback (Kulsum et al., 2024), while PatchIsland studies continuous repair through agent ensembles, deduplication, and validation (Kim et al., 2026). RvB adds an active attacker that probes every updated artifact: accepted exploits condition the next repair, and post-update vulnerability plus regression checks connect that repair to security and utility outcomes.

Adaptive red-teaming and guardrails. Automated jailbreakers search over attack principles, reasoning traces, reusable strategies, or multi-turn dialogues (Xiong et al., 2025a; Ying et al., 2025; Liu et al., 2025; Rahman et al., 2025). Defenses optimize prompts, maintain dynamic policies, or retrieve prior attacks (Xiong et al., 2025b; Hoover et al., 2025; Yang et al., 2025a). MART alternates automatic red-teaming with safety fine-tuning (Ge et al., 2024); concurrent CHASE similarly co-trains attacker and defender with reinforcement learning (Markasserithodi et al., 2026). RvB occupies a complementary test-time setting: model weights remain fixed, while guardrail rules or program code are the objects that carry the updates.

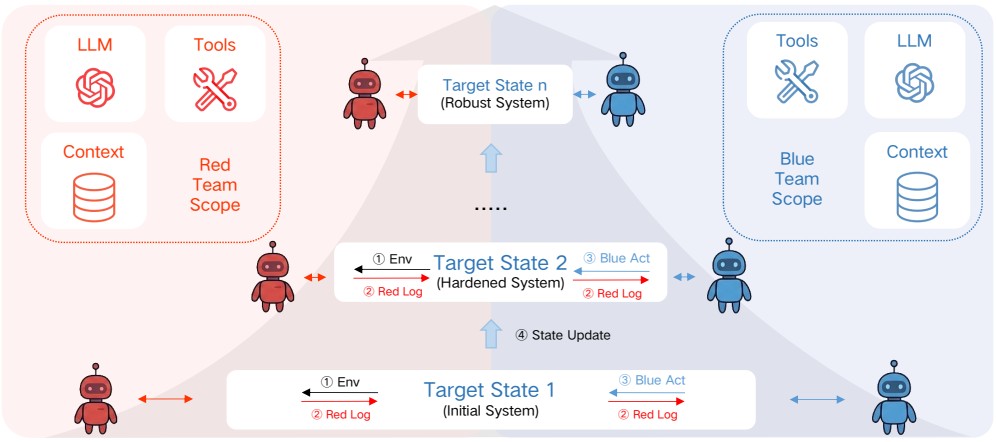


Figure 2: RvB alternates attack discovery and remediation while the LLMs and tools remain fixed. Code updates receive post-update vulnerability and regression checks; guard updates receive a benign-input screen, are attacked in the next round, and are later diagnosed on retained attacks.

Positioning. Across both domains, RvB's distinguishing unit is the *updated artifact under renewed attack*. Unlike one-shot red teaming, a successful attack changes what the defender edits next. Unlike cooperative repair, the attacker returns after each edit. Unlike MART and CHASE, adaptation is carried by code or rules rather than fine-tuned weights. This positioning motivates a shared protocol while leaving each domain's validator and failure modes explicit.

3 Stateful Red–Blue Feedback

Interaction structure. RvB is a finite-horizon, sequential, partially observed interaction between fixed Red and Blue policies over a persistent artifact. At round k, $S_k = (X_k, C_k)$ contains the current artifact X_k and an archive C_k. Each policy receives a task-configured view, $\phi_R(S_k)$ or $\phi_B(S_k, E_k)$, rather than necessarily observing the full archive. With fixed agents R_{θ_R} and B_{θ_B}, one round is

$$A_k = R_{\theta_R}(\phi_R(S_k)), \tag{1}$$
$$E_k = \{a \in A_k : V_{\text{atk}}(a, S_k) = 1\}, \tag{2}$$
$$\Delta_k = B_{\theta_B}(\phi_B(S_k, E_k)), \tag{3}$$
$$S_{k+1} = T(S_k, \Delta_k), \tag{4}$$
$$O_{k+1} = (o_{\text{sec},k+1}, o_{\text{util},k+1}). \tag{5}$$

Here V_{atk} is an executable check for code or a model-based judge for guardrails. Training-free means that θ_R and θ_B remain fixed; T carries the effects of earlier rounds through program code, deployment state, or guardrail rules.

Persistent state and evidence. X_k carries the editable source tree and service, or external guardrail rules; C_k is an audit archive. The archive need not be returned verbatim to either agent. Red probes the current artifact, Blue receives accepted attacks and edit access, and the next Red call probes the updated artifact. The three commitments are accepted attacks only, persistent updates, and joint security–utility observations. They do not impose a shared validator, payoff, or rollback rule.

Evaluation timing. Code patches receive post-update vulnerability and regression checks. Guardrail updates receive a benign-input screen, face attacks in the next Red round, and are evaluated separately on retained attacks. Thus an update is an empirical test, not a guaranteed improvement. Table 9 aligns the two domains; Appendix D details the state, memory, and stopping conventions.

4 Threat Model and Evaluation Contract

RvB targets local, testable artifacts. For code, Red accesses the deployed service and permitted shell/MCP tools, while Blue receives a successful-attack report and source access. For guardrails, Red queries the guarded target under a nominal attempt budget and Blue edits only external rules. Model weights remain fixed.

An attack counts only when the executable check or model-based judge accepts it on the current state. Code defense requires attack blocking together with a passing regression check; guardrail updates are screened on benign prompts. These observations do not certify untested endpoints, benign distributions, or attack families.

The cooperative code reference uses the same nominal agents, tools, and model settings, but its cumulative-union estimator differs from RvB's final-state estimator. We therefore treat it as a descriptive reference, not a matched estimate of the protocol effect. The design does not isolate report format, retries, or test coverage.

5 Experimental Evaluation

We study code hardening and guardrail optimization. Figures 3 and 7 show their respective loops; Appendices A–D provide budgets, diagnostics, and evidence details.

5.1 Code Hardening

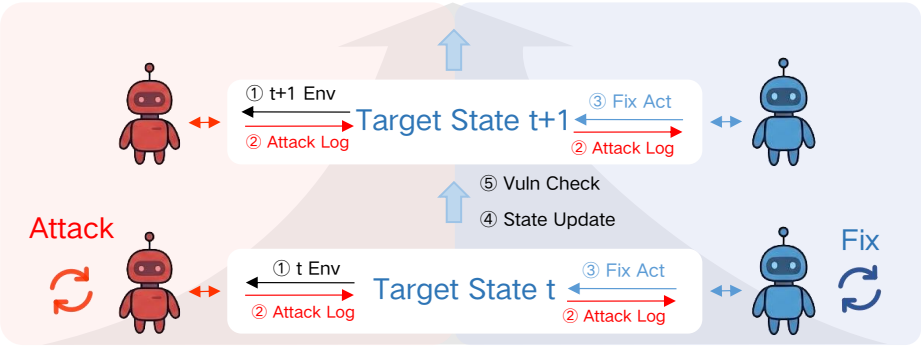


Figure 3: Code-hardening loop. Reproduced exploits guide persistent source updates, followed by vulnerability and service-regression checks.

Setup and protocol. We deploy two vulnerable PHP applications in isolated Docker services: *Pharmacy Management System v1.0* and *OOP CMS BLOG 1.0*. CAI (Mayoral-Vilches et al., 2025) probes the service through Bash and MCP tools and reports an exploit only after reproduction. Mini-SWE-Agent (Yang et al., 2024) receives the affected file, source span, root cause, and reproduction evidence; it localizes the fault, patches code, restarts the service, and runs vulnerability and regression checks. Each Red call permits 30 turns, a failed repair allows up to three refinements, and the outer loop runs for at most five epochs. Early stopping follows null outputs, batch failure, or three epochs with unchanged vulnerability counts. The manuscript identifies the default backbone as `gemini-3.0-flash`, without an exact provider version; Qwen3-Max is a separate backbone record.

CAI separates planning, execution, and reporting: reconnaissance and service feedback inform attack hypotheses, tool calls test them, and the reporter records successful reproductions. Blue then uses this localized evidence to inspect the relevant source before changing the application. The source tree and deployed service persist across epochs, so subsequent probing encounters the effects of earlier patches. This closes the loop between discovery and remediation while retaining separate checks for exploit blocking and service functionality; it also makes the sequence of artifacts, rather than a collection of independent repair attempts, the object of evaluation.

Metrics. For check i, $r_{\text{atk}}^{(i)} = 1$ denotes a successful exploit and $r_{\text{reg}}^{(i)} = 1$ a passing regression check. With N checks,

$$\text{TDSR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[r_{\text{atk}}^{(i)} = 0 \wedge r_{\text{reg}}^{(i)} = 1], \tag{6}$$

$$\text{FDSR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[r_{\text{atk}}^{(i)} = 0], \tag{7}$$

$$\text{SDR} = \text{FDSR} - \text{TDSR}. \tag{8}$$

The reported code DSR is TDSR. The manuscript does not specify whether the check population and its size N stay fixed across rounds; ASC alone does not determine this denominator. Attack Success Count is $\text{ASC}_k = |\bigcup_{t \le k} \mathcal{E}_t^{\text{succ}}|$, the cumulative number of distinct reproduced exploits, where $\mathcal{E}_t^{\text{succ}}$ is the set of reproduced exploits found in round t. ASC measures coverage, not attack sophistication; FDSR alone may include an unavailable service.

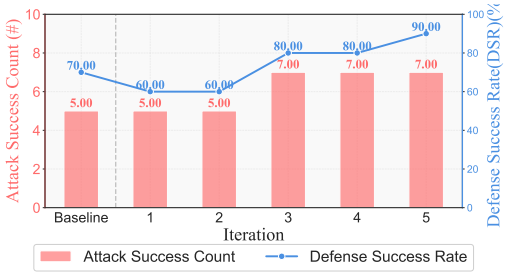


Figure 4: Pharmacy trajectory. Cumulative reproduced exploits rise from five to seven, while DSR reaches 90%.

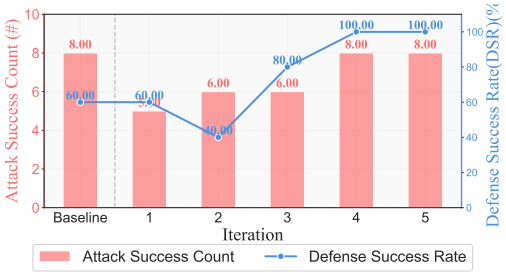


Figure 5: OOP CMS trajectory. DSR falls from 60% to 40% before recovering to 100%; ASC reaches eight.

Pharmacy: defensive progress with broader exploit coverage. Figure 4 tracks both the current defensive outcome and the accumulated set of reproduced exploits. Pharmacy begins at 60% DSR, reaches 80% at the third iteration, and finishes at 90%. Over the same trajectory, ASC increases from five to seven. Thus the recorded defense improves while Red continues to discover additional vulnerabilities in the evolving service. The cooperative summary reports 70% DSR and five exploits. Its DSR lies above the initial RvB value and below the later values, but this comparison is descriptive because the two methods use different aggregation conventions. The increase in ASC establishes broader reproduced-exploit coverage; it does not by itself establish more sophisticated attacks.

OOP CMS: an intermediate regression before recovery. The second application exhibits a different path to its endpoint (Figure 5). Its five-round DSR sequence is $60 \to 40 \to 80 \to 100 \to 100\%$, while ASC follows $5 \to 6 \to 6 \to 8 \to 8$. The second round therefore combines one additional reproduced exploit with a lower defense score. Subsequent rounds recover, reaching eight cumulative exploits and 100% DSR by Round 4. This trajectory motivates evaluating each updated artifact: the endpoint alone would hide the early decline. Because the manuscript does not state whether the check population stays fixed across rounds, the decline cannot be attributed specifically to a faulty patch or to newly discovered weaknesses. The final plateau describes the recorded checks, not exhaustion of the application's attack surface.

Backbone sensitivity. The separate Qwen3-Max record has DSR $30 \to 50 \to 60 \to 60 \to 60\%$ and ASC $1 \to 5 \to 7 \to 7 \to 7$ (Table 4). Both quantities stop changing after Round 3. Relative to its cooperative summary of 50% DSR and five exploits, the final state has higher reported defense and coverage, yet further iterations produce no recorded gain. Together with the OOP CMS decline, this plateau shows why additional rounds should be evaluated rather than assumed beneficial. The application used for Qwen3-Max is not specified in the manuscript, so this record supports executability with another backbone, not two-environment evidence under Qwen.

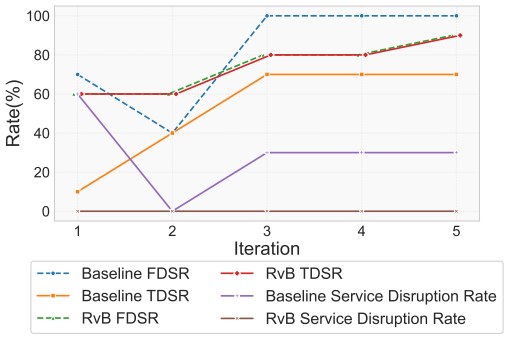


Figure 6: Pharmacy utility diagnostic. RvB's FDSR and TDSR coincide in all rounds; the cooperative SDR reaches 60%.

Setting	Method	ASC	DSR
Pharmacy	Coop.	5	70%
Pharmacy	RvB	7	90%
OOP CMS	Coop.	8	60%
OOP CMS	RvB	8	100%
Qwen3-Max	Coop.	5	50%
Qwen3-Max	RvB	7	60%

Table 1: Code outcomes. Pharmacy and OOP CMS use the recorded Gemini backbone. Coop. aggregates a five-epoch union; RvB reports the final state.

Service-preserving remediation. Figure 6 separates blocking an exploit from repairing a usable service. An unsuccessful exploit is insufficient evidence of a valid fix if the patch also disables the affected interface or breaks a regression check. In the Pharmacy RvB trajectory, FDSR and TDSR coincide at every round, yielding zero measured SDR throughout. In the cooperative diagnostic, the gap reaches 60 percentage points, indicating that attack blocking does not always coincide with passing regression checks. This distinction explains why TDSR is the primary code-defense outcome. The structured reproduction report supplies an affected file, source span, and root cause to guide a targeted repair; the utility checks then test whether the resulting service remains functional. The observed alignment is consistent with that design, but without component ablations it does not isolate the causal contribution of the report or validator.

Comparison and accounting. Table 1 summarizes the available endpoints. The cooperative workflow aggregates the five-epoch union, whereas RvB reports the final artifact. A union of fixes found across multiple artifacts need not describe the security of any single deployed artifact. Conversely, a final artifact can retain earlier repairs while being tested against a changed set of exploits. These summaries therefore cannot be converted to a common estimator without the per-epoch artifacts and checks. Zero measured SDR means no evaluated check combined a blocked exploit with a failed regression test. Table 5 reports token accounting; its "Red only" row is the offense-only setting without Blue remediation.

5.2 Guardrail Optimization

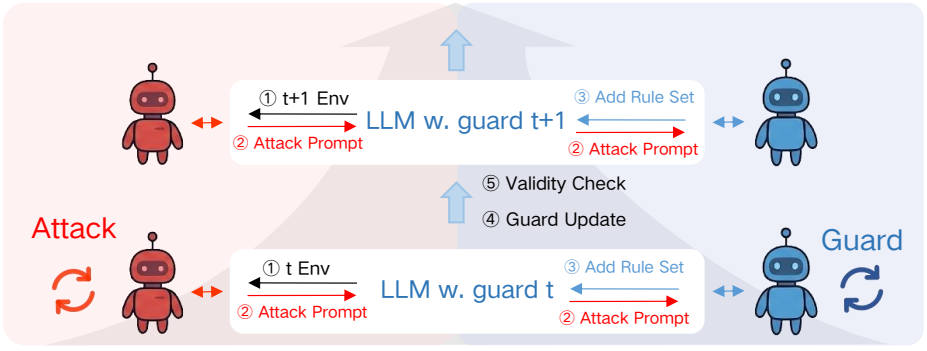


Figure 7: Guardrail loop. Accepted jailbreaks guide persistent rule updates, which receive benign-input screening before renewed attacks.

Setup and procedure. Red attacks HarmBench targets (Mazeika et al., 2024) with Composition-of-Principles (Xiong et al., 2025a); Blue revises a NeMo-based guardrail (Rebedea et al., 2023).

The protected model is Qwen2-7B-Instruct. DeepSeek-V3 generates strategies and benign prompts; Qwen3-32B instantiates and judges attacks; GPT-4o mini classifies inputs and patches rules. Per harmful target, Red has three strategy-refinement rounds of up to ten attempts each. Blue screens updates with three generated benign prompts per harmful target, then the next Red round attacks the persistent rules. Retained attacks are replayed separately. The manuscript does not state whether every triggering jailbreak is replayed immediately after its update. Attack generation and judging use the same model family.

The editable object is the external rule set surrounding the protected model. A successful jailbreak supplies evidence for a rule revision, and Blue attempts to abstract patterns from that evidence rather than store only the triggering prompt. The next Red call then probes the revised guard, creating an adaptive sequence of attack targets for the defender. This design distinguishes improvements against newly generated attacks from retention against stored attacks; the two are evaluated separately below. Whether a proposed rule preserves useful access is checked through benign inputs rather than inferred from harmful-input blocking alone.

Metrics. For harmful targets $\mathcal{D}_{\text{harm}}$, let $\mathcal{S}_i$ be those successfully attacked in round i and $k_{q,i}$ the attempts used. Current-round defense and successful-attack effort are

$$\text{DSR}_i = 1 - \frac{|\mathcal{S}_i|}{|\mathcal{D}_{\text{harm}}|}, \qquad \text{AAT}_i = \frac{\sum_{q \in \mathcal{S}_i} k_{q,i}}{|\mathcal{S}_i|}. \tag{9}$$

AAT is defined only when $|\mathcal{S}_i| > 0$ and is conditioned on success. CRDE measures blocking on each fixed retained attack set $\mathcal{A}_j$ under later guards; FPR measures blocking on round-specific generated benign sets. Their equations appear in Appendix B.

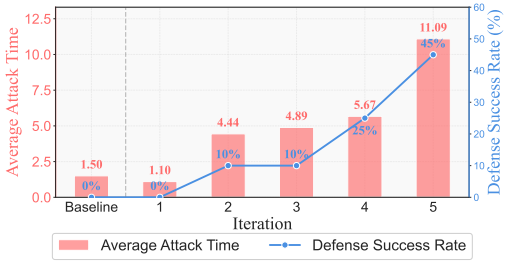


Figure 8: Current-round DSR and AAT. The original "Average Attack Time" axis counts attempts among successful attacks, not elapsed time.

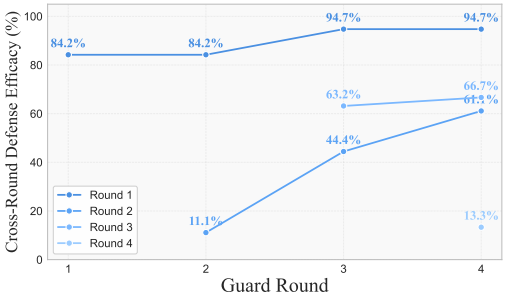


Figure 9: Historical-attack replay. Each curve evaluates one fixed retained attack set against later guard states.

Security trajectory and attack effort. Across baseline and five updates, DSR is 0%, 0%, 10%, 10%, 25%, and 45%; AAT is 1.50, 1.10, 4.44, 4.89, 5.67, and 11.09 (Figure 8). The endpoint therefore combines a higher fraction of defended targets with approximately 7.4 times the baseline number of attempts among attacks that still succeed. The early AAT decrease and DSR plateaus rule out strictly monotone progress. Moreover, the successful subset changes as defense improves, so an AAT increase is not a paired estimate of additional effort on every target. At the endpoint, 55% of harmful targets remain successfully attacked, leaving substantial headroom for further hardening without establishing that additional rounds would close this gap.

Protection against retained attacks. CRDE asks whether later guards preserve protection against a fixed set of earlier attacks, a question that current-round DSR alone cannot answer (Figure 9). Round-1 efficacy rises from 84.2% to 94.7%; the Round-2 set follows 11.1%, 44.4%, and 61.1%; and the Round-3 set follows 63.2% and 66.7%. No plotted within-set trajectory decreases. The Round-2 pattern is particularly informative: attacks that initially remain effective are blocked more often after later updates, consistent with rule additions covering retained weaknesses. Comparisons must stay within each fixed attack set. In particular, 63.2% for the Round-3 set and 61.1% for the Round-2 set do not establish a temporal regression. These replay results support retention on the

stored attacks, while untested paraphrases, alternative strategies, and changes in benign behavior remain outside this diagnostic.

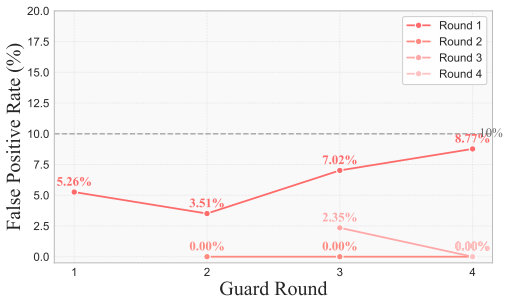


Figure 10: False positives on round-specific benign sets. Observed values range from 0% to 8.77%, below the plotted 10% reference.

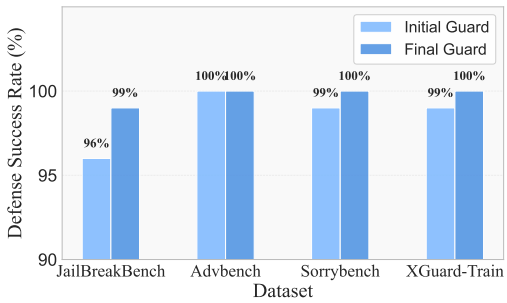


Figure 11: External-set check. Initial scores are 96–100%; endpoint gains range from zero to three percentage points.

Benign-input utility. The FPR diagnostic complements harmful-input evaluation by measuring how often benign prompts are blocked (Figure 10). Across the plotted round-specific sets, observed FPR ranges from 0% to 8.77%, and all values remain below the 10% reference line. Some sets exhibit zero measured false positives, but this is not a uniform property of all evaluated sets or rounds. These observations support the narrower conclusion that the reported defensive gains coexist with limited measured over-interception on the available benign prompts. The procedure generates three benign prompts per harmful target for validation. The FPR definition evaluates each round-indexed benign set against a specified guard state. The manuscript does not specify whether validation prompts were reused for all later FPR measurements, nor does it provide exact set sizes. Thus the plotted percentages alone do not establish exact false-positive counts or uncertainty estimates. Neither check establishes utility on broader production traffic.

External-set transfer. The initial-to-final results are JailbreakBench (Chao et al., 2024) $96 \rightarrow 99\%$, AdvBench (Zou et al., 2023) $100 \rightarrow 100\%$, SORRY-Bench (Xie et al., 2024) $99 \rightarrow 100\%$, and XGuard-Train (Upadhayay & Behzadan, 2025) $99 \rightarrow 100\%$ (Figure 11). Thus three sets improve and one remains unchanged. The largest absolute gain is three percentage points on JailbreakBench; the already-perfect AdvBench score provides a no-regression check rather than an improvement. These high initial scores also explain why the external results should be interpreted separately from the more challenging adaptive-attack trajectory. The reported evaluation uses 100 sampled prompts from four external sets, but the manuscript does not specify their per-set allocation or provide a prompt-overlap audit. The observations are consistent with transfer beyond the interaction prompts, but support modest measured gains rather than a broad claim of generalization to unseen attacks.

Record alignment. Current-round DSR, success-conditioned AAT, retained-set CRDE, and benign-set FPR answer different questions. CRDE and token records use four guard states/rounds, whereas DSR and AAT show a baseline plus five updates; no one-to-one mapping is inferred. The task grid (Figure 12) remains in the appendix because entries 34 and 39 exceed the nominal 3×10 cap and their logging unit is unresolved.

5.3 Validity and Scope

The manuscript reports trajectories without specifying the number of independent runs or providing uncertainty estimates. The evaluated scope is two PHP services, one protected language model, and one principal jailbreak method. The final guard still admits successful attacks on 55% of harmful targets, and Qwen TDSR leaves 40% of checks outside the joint blocking-and-regression-pass outcome; aggregates do not identify failure categories. The manuscript does not provide exact model/API revisions, decoding settings, some denominators, or per-attempt logs. Results support feasibility and trajectory analysis, not statistical rankings or broad generalization.

6 Discussion and Limitations

Artifact-carried adaptation. RvB's common mechanism is an evidence-conditioned update to a persistent artifact, followed by renewed security and utility observations. The artifact carries adaptation while model weights remain fixed. Code and guardrails share this structure, but their validators, evaluation timing, and DSR definitions differ (Table 9).

Why trajectories matter. The OOP CMS regression and Qwen plateau show why a final score alone is incomplete. Guardrail CRDE instead improves or plateaus within each plotted retained set; it does not establish a replay regression. Neither pattern guarantees behavior against untested attacks or benign traffic.

Feedback as an adaptive source of examples. Renewed attacks expose weaknesses of the current artifact, so each successful reproduction or accepted jailbreak supplies an example tied to the defender's present state. This is the practical motivation for an iterative interaction: after an update, Red tests the remaining attack surface instead of repeatedly presenting an unchanged challenge. The code coverage increase and guardrail replay improvements are consistent with this mechanism. However, they do not distinguish a general defensive principle from several targeted rules, nor do they establish convergence to an equilibrium. The framework provides an operational way to generate and test candidate improvements; the strength of each conclusion depends on the observations collected after the update.

Security and utility require separate observations. The code diagnostic illustrates why attack failure and service preservation must be recorded together. Guardrail evaluation has the analogous concern that broad rejection can block harmful requests while also obstructing benign ones. TDSR and SDR address this issue through regression checks, whereas guardrail FPR addresses it through benign prompts. Their shared purpose motivates the two-observation interface in Section 3, but the checks remain domain-specific and incomplete. A stronger security result on one set therefore needs to be read alongside its corresponding utility evidence, not as a substitute for it.

Evidence still needed. Matched comparisons must align initial artifacts, vulnerability inventories, models, tools, retry and query budgets, and final-state estimators; Appendix D details the required records. Repeated trials, state-reset and evidence/validator ablations, additional targets, and independent judges remain necessary to isolate mechanism and reliability. The present cases do not establish which component causes the observed gains or provide a general security certificate.

Limits of the turn-based setting. The experiments use a controlled sequence of attack, update, and evaluation steps. This abstraction simplifies asynchronous security operations, where attacks and legitimate requests may continue while a repair is being prepared or deployed. Outcomes also depend on the reasoning and tool-use capabilities of the fixed agents: persistence cannot compensate for a vulnerability Red never discovers or a repair Blue cannot implement. Consequently, a stable score within the scheduled rounds should be interpreted relative to the available agents, tools, and the coverage of the security and utility checks.

7 Conclusion

RvB organizes fixed Red and Blue agents around a persistent artifact: retain checker- or judge-accepted attacks, update the current code or rules, observe security and utility, and attack the resulting state again. The code workflow combines reproduced exploits with patching and available post-update vulnerability and regression checks, while the guardrail case combines renewed probing with retained-attack and benign-input evaluation. The code regression and observed plateaus demonstrate why hardening should be evaluated as a trajectory rather than only by its endpoint. RvB makes the artifact state, accepted attacks, update transition, and task-specific observations explicit, connecting active red teaming to automated remediation without model-weight updates.

Future work will extend evaluation to more diverse benchmarks and investigate multimodal environments and longer-horizon security tasks. These directions will test the scope of the framework beyond the applications and interaction schedules evaluated here.

AI USE STATEMENT

Generative AI tools assisted with language polishing, grammatical corrections, LaTeX formatting, and checks for consistency between the manuscript text and existing figures. The authors take responsibility for the final manuscript. The LLM components used in the experiments are described in Section 5.

ETHICS STATEMENT

RvB is dual use. Code experiments used isolated, network-gated Docker services built from public vulnerable applications; no production or externally hosted service was probed. Red-team tools stayed within the local loop, and candidate defenses received the available vulnerability, regression, or benign-input checks.

REPRODUCIBILITY STATEMENT

Section 3 specifies the interaction structure, and Section 5 defines the evaluation metrics and reports the available configurations. Appendices A and B provide role and budget details, aggregate diagnostics, and token accounting. The manuscript reports configurations and aggregate results but does not specify all model snapshots, decoding settings, or per-attempt records. Anonymous project links: code hardening, guardrails, and red teaming.

REFERENCES

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

Sebastián R. Castro, Roberto Campbell, Nancy Lau, Octavio Villalobos, Jiaqi Duan, and Alvaro A. Cardenas. Large language models are autonomous cyber defenders. In *2025 IEEE Conference on Artificial Intelligence (CAI)*, pp. 1125–1132. IEEE, May 2025. doi: 10.1109/cai64502.2025.00195. URL http://dx.doi.org/10.1109/CAI64502.2025.00195.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramèr, Hamed Hassani, and Eric Wong. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/63092d79154adebd7305dfd498cbff70-Abstract-Datasets_and_Benchmarks_Track.html.

David de Fitero-Dominguez, Eva Garcia-Lopez, Antonio Garcia-Cabot, and Jose-Javier Martinez-Herraiz. Enhanced automated code vulnerability repair using large language models. *Engineering Applications of Artificial Intelligence*, 138:109291, December 2024. ISSN 0952-1976. doi: 10.1016/j.engappai.2024.109291. URL http://dx.doi.org/10.1016/j.engappai.2024.109291.

Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. Pentestgpt: An llm-empowered automatic penetration testing tool, 2024. URL https://arxiv.org/abs/2308.06782.

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models, 2023. URL https://arxiv.org/abs/2212.14834.

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

Suyu Ge, Chunting Zhou, Rui Hou, Madian Khabsa, Yi-Chia Wang, Qifan Wang, Jiawei Han, and Yuning Mao. MART: Improving LLM safety with multi-round automatic red-teaming. In *Proceedings of the 2024 Conference of the North American Chapter of the Association*

for Computational Linguistics: Human Language Technologies, pp. 1927–1937, 2024. doi: 10.18653/v1/2024.naacl-long.107.

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

Julius Henke. Autopentest: Enhancing vulnerability management with autonomous llm agents, 2025. URL https://arxiv.org/abs/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. Dynaguard: A dynamic guardian model with user-defined policies, 2025. URL https://arxiv.org/abs/2509.02563.

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

Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms, 2023. URL https://arxiv.org/abs/2303.07263.

Wonyoung Kim, Seunggi Min, Minjae Gwon, Dowoo Baik, Haein Lee, Hyeon Heo, Minjae Lee, Min Woo Baek, Yonghwi Jin, Younggi Park, Yunjae Choi, Taesoo Kim, Sangdon Park, and Insu Yun. Patchisland: Orchestration of LLM agents for continuous vulnerability repair, 2026.

Ummay Kulsum, Haotian Zhu, Bowen Xu, and Marcelo d'Amorim. A case study of LLM for automated vulnerability repair: Assessing impact of reasoning and patch validation feedback. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, 2024. doi: 10.1145/3664646.3664770.

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

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

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

Rahul Markasserithodi, Aditya Joshi, Yuekang Li, Ishmanbir Singh, Chris Yoo, and Alan Niu. CHASE: Adversarial red–blue teaming for improving LLM safety using reinforcement learning, 2026.

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. Cai: An open, bug bounty-ready cybersecurity ai, 2025. URL https://arxiv.org/abs/2504.06017.

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

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

Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. 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*, pp. 431–445. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-demo.40. URL https://aclanthology.org/2023.emnlp-demo.40/.

Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. Pentestagent: Incorporating llm agents to automated penetration testing, 2025. URL https://arxiv.org/abs/2411.05185.

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

Bibek Upadhayay and Vahid Behzadan. X-Guard: Multilingual guard agent for content moderation. *arXiv preprint arXiv:2504.08848*, 2025. URL https://arxiv.org/abs/2504.08848.

Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, pp. 1–13. ACM, April 2024. doi: 10.1145/3597503.3639121. URL http://dx.doi.org/10.1145/3597503.3639121.

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

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

Chen Xiong, Xiangyu Qi, Pin-Yu Chen, and Tsung-Yi Ho. Defensive prompt patch: A robust and interpretable defense of llms against jailbreak attacks, 2025b. URL https://arxiv.org/abs/2405.20099.

Guangyu Yang, Jinghong Chen, Jingbiao Mei, Weizhe Lin, and Bill Byrne. Retrieval-augmented defense: Adaptive and controllable jailbreak prevention for large language models, 2025a. URL https://arxiv.org/abs/2508.16406.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

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

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

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

A.1 AGENT ROLES AND BUDGETS

Red Team. The CAI agent separates planning, execution, and reporting. It first performs passive reconnaissance, then proposes and executes an attack path through its permitted Bash and MCP interfaces. A structured report is produced after a successful exploit is recorded on the current service. The report contains the affected file and code span, a root-cause description, and reproduction evidence. The maximum interaction budget is 30 turns per Red call.

Blue Team. Mini-SWE-Agent receives the report and the repository. It performs fault localization, patch generation, application, service restart, vulnerability checking, and regression checking. If the initial repair fails, the agent may make at most three retry attempts. The default backbone for both agents is `gemini-3.0-flash` in the manuscript, which does not specify the provider endpoint or exact model version. The reported backbone ablation uses Qwen3-Max with the same nominal agent and tool settings.

Outer loop. The stopping rule uses $max_epoch = 5$ and terminates earlier on null production by both agents, batch-execution failure, or an unchanged vulnerability count for three consecutive epochs. The experiment archives interaction artifacts and the vulnerability-reduction trajectory for each epoch according to the experiment description. The manuscript presents aggregate results rather than the full set of run-level records.

A.2 ROUND-LEVEL EVIDENCE RECORD

The paper reports one representative code record with an affected file, source span, root-cause analysis, and reproduction evidence. Together with the stated Blue workflow, it motivates the summary below. The first three rows describe the example attack report; the remaining rows describe subsequent workflow steps, not additional fields shown in that report:

Field / step	Reported content or workflow
file / span	Affected path and code region; e.g., php_action/removeOrder.php.
root cause	Failure mechanism used for localization; the representative record identifies unsanitized interpolation of an identifier into a query.
reproduction	Request or payload that reproduced the issue on the current local service, as illustrated by the representative attack report.
patch	Source diff associated with the accepted attack in the stated workflow.
restart	Service restart after applying the patch to the Docker environment.
vulnerability check	Reported mitigation outcome after the update.
regression	Result of the available service or functional check.

Table 2: Representative evidence schema derived from the reported record and workflow; this is not a per-attack completeness claim.

This schema expresses the intended link from an attack accepted on S_k to a candidate transition and post-update observations on S_{k+1}. The available aggregate results remain bounded by the reported vulnerability and regression coverage.

A.3 Outcome Definitions

For vulnerability check i, $r^{(i)}_{\text{atk}} = 1$ denotes a successful exploit and $r^{(i)}_{\text{reg}} = 1$ denotes a passing regression test. With N checks,

$$\text{TDSR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[r^{(i)}_{\text{atk}} = 0 \wedge r^{(i)}_{\text{reg}} = 1], \tag{10}$$

$$\text{FDSR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[r^{(i)}_{\text{atk}} = 0], \tag{11}$$

$$\text{SDR} = \text{FDSR} - \text{TDSR}. \tag{12}$$

Throughout the code study, the reported DSR denotes TDSR. FDSR is therefore an apparent attack-failure rate that includes both valid repairs and outage-induced failures. It should not be described as the rate of destructive patches. TDSR isolates the cases in which attack blocking and measured service preservation hold together.

Let $\mathcal{E}_k$ be the set of unique reproduced exploits retained through round k. Attack Success Count is

$$\text{ASC}_k = |\mathcal{E}_k|, \qquad \mathcal{E}_k = \bigcup_{t=1}^{k} \mathcal{E}^{\text{succ}}_t, \tag{13}$$

where $\mathcal{E}^{\text{succ}}_t$ contains the reproduced exploits found in round t. ASC measures reproduced exploit coverage. It is not a calibrated measure of attack complexity or sophistication.

A.4 Per-Round Diagnostics

Figure 5 visualizes the OOP CMS record in the main text. The tables below retain its exact values and the Qwen3-Max trajectory.

Metric	Coop.	R1	R2	R3	R4	R5
ASC	8	5	6	6	8	8
DSR	60%	60%	40%	80%	100%	100%

Table 3: OOP CMS BLOG 1.0. “Coop.” is the cumulative-union cooperative summary; R1–R5 are the RvB trajectory.

Metric	Coop.	R1	R2	R3	R4	R5
ASC	5	1	5	7	7	7
DSR	50%	30%	50%	60%	60%	60%

Table 4: Qwen3-Max code record. “Coop.” is the cumulative-union cooperative summary; R1–R5 are the RvB trajectory.

Table 3 shows a non-monotone trajectory: DSR decreases at Round 2 before recovering. Table 4 shows a different limit: the endpoint exceeds the baseline value, but DSR plateaus after Round 3. These trajectories demonstrate why progress must be evaluated round by round.

A.5 Utility-Aware Code Diagnostic

Figure 6 reports the available Pharmacy utility diagnostic. RvB has zero measured SDR in all five rounds, whereas the cooperative gap reaches 60%.

A.6 Code Token Accounting

Code setting	Input	Output	Total
Red only	576,140	17,742	593,882
Cooperative	5,943,099	310,093	6,253,192
RvB	4,800,430	305,329	5,105,759

Table 5: Code-hardening token totals over the five-epoch setting.

"Red only" denotes the offense-only setting without Blue remediation. The reported totals are 6,253,192 tokens for the cooperative workflow and 5,105,759 for RvB, an 18.35% reduction over the five-epoch setting. These token counts are descriptive accounting measurements. They do not establish runtime, monetary cost, or asymptotic complexity because these also depend on provider pricing, hardware, latency, and caching, which the token totals do not determine.

B Guardrail-Optimization Protocol

B.1 Components and Interaction

Component	Role	Reported model
Target	protected model	Qwen2-7B-Instruct
Red	strategy helper	DeepSeek-V3
Red	attack instantiation	Qwen3-32B
Evaluator	attack judge	Qwen3-32B
Blue	guard classification	GPT-4o mini
Blue	rule patching	GPT-4o mini
Validation	benign generation	DeepSeek-V3

Table 6: Model-family roles reported for the guardrail experiment. The manuscript names these models but does not specify exact checkpoint revisions or API snapshots, generation modes, or decoding settings.

For each harmful target, the Red Team receives up to three strategy-refinement rounds and up to ten attack attempts per refinement round. Successful prompts are passed to the Blue Team, which abstracts shared patterns and proposes rule-set changes. Three generated benign prompts per harmful target are used as the immediate post-update utility screen. Security is measured when the next Red round targets the updated guard, while retained attacks are used in the reported CRDE diagnostics. The manuscript does not state whether every triggering jailbreak is replayed immediately after each update.

B.2 Metrics

Let $\mathcal{D}_{\text{harm}}$ be the harmful targets, $\mathcal{S}_i$ the subset successfully attacked in round i, $k_{q,i}$ the attempts used for target q, $p_{q,i}$ its attack prompt, $\mathcal{A}_j$ the attacks retained from round j, and G_i the guard after round i:

$$\text{DSR}_i = 1 - \frac{1}{|\mathcal{D}_{\text{harm}}|} \sum_{q \in \mathcal{D}_{\text{harm}}} \text{IsSuccess}(p_{q,i}, G_{i-1}), \tag{14}$$

$$\text{AAT}_i = \frac{1}{|\mathcal{S}_i|} \sum_{q \in \mathcal{S}_i} k_{q,i}, \tag{15}$$

$$\text{CRDE}_{i,j} = \frac{1}{|\mathcal{A}_j|} \sum_{p \in \mathcal{A}_j} \left(1 - \text{IsSuccess}(p, G_i)\right). \tag{16}$$

For a benign set $\mathcal{D}_{\text{benign},j}$,

$$\text{FPR}_{i,j} = \frac{1}{|\mathcal{D}_{\text{benign},j}|} \sum_{p_b \in \mathcal{D}_{\text{benign},j}} \text{IsBlocked}(p_b, G_i). \tag{17}$$

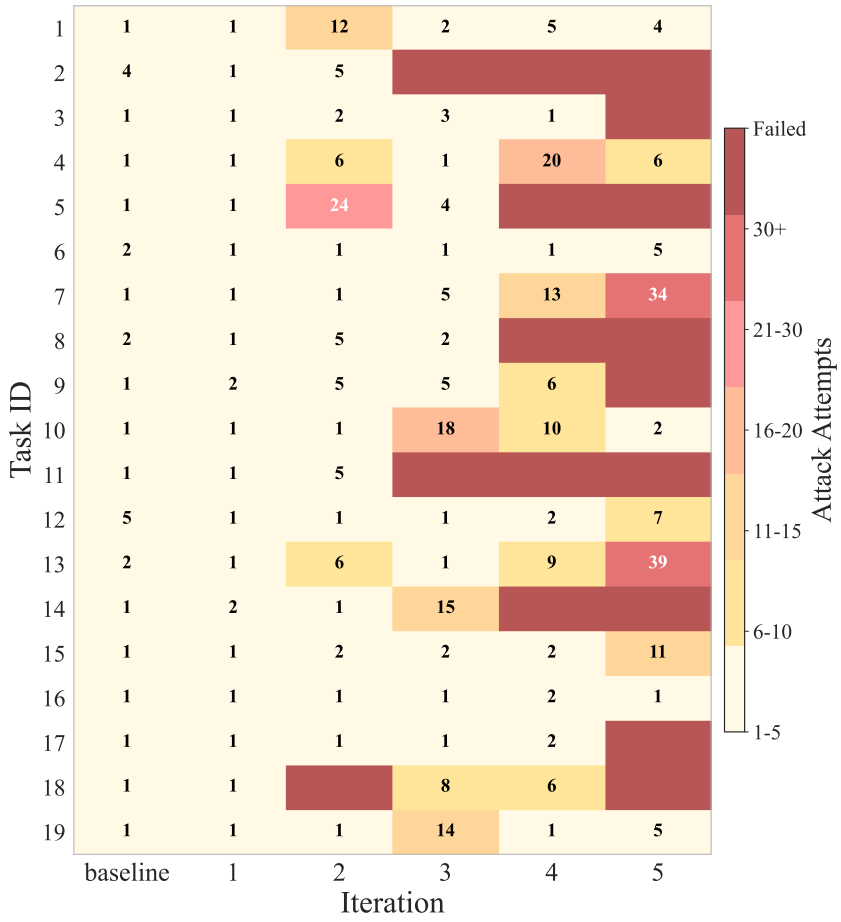


Figure 12: Recorded task-level attack-attempt grid. Unnumbered dark-red cells denote failed attacks. Because 34 and 39 exceed the nominal 3×10 attempt cap, this diagnostic is retained for traceability but not used to support a main-text quantitative claim.

AAT is defined only when $|\mathcal{S}_i| > 0$ (otherwise it is N/A) and is conditioned on attack success. An increase can therefore reflect harder surviving attacks as well as genuine broad hardening; it must be interpreted jointly with DSR and the task grid.

B.3 Retained-Attack and Benign-Input Diagnostics

Figures 9 and 10 show retained-attack replay and benign-input false positives, respectively. Each CRDE curve corresponds to a fixed retained attack set and should be read separately.

The task grid marks failed attacks with unnumbered dark-red cells, matching the "Failed" category in its legend. The numerical entries 34 and 39 exceed the stated 3×10 attempt budget; the manuscript does not explain this discrepancy. These counts and the failed-attack cells are distinct annotations.

B.4 External-Benchmark and Benign Checks

Dataset	Initial	Final	Δ
JailbreakBench	96%	99%	+3
AdvBench	100%	100%	0
SORRY-Bench	99%	100%	+1
XGuard-Train	99%	100%	+1

Table 7: Defense success in the reported external-set check. The source states that 100 prompts were sampled from four sets but leaves total-versus-per-set count and allocation ambiguous.

The final guard does not regress on the four reported external prompt sets, but the baseline is already between 96% and 100%. Table 7 therefore supports only a no-regression result with modest gains. Across the round-specific benign sets, all measured FPR values fall below the 10% reference line shown in the source figure; the largest observed value is 8.77%. These checks do not substitute for production traffic evaluation or human preference judgments. The described attack generator and judge both use Qwen3-32B; the manuscript does not separately specify the evaluator configuration for the external-set check. It also does not report exact checkpoint/API versions, decoding settings, per-set denominators, sampling allocation, or a prompt-level overlap audit.

B.5 Guardrail Token Accounting

Guard setting	Total tokens
Baseline	4,040,889
RvB Round 1	2,828,789
RvB Round 2	5,961,159
RvB Round 3	10,490,319
RvB Round 4	12,223,668

Table 8: Recorded guardrail token totals. These per-round totals are not cumulative across rows.

The rows are recorded per-setting measurements and are not cumulative. They do not establish runtime or monetary cost.

The guardrail records do not fully align their sequence labels: token and CRDE tables name four RvB rounds/guard states, while DSR, AAT, and the task grid show a baseline plus five reported iterations. We retain those labels without assuming a one-to-one mapping.

C Cross-Domain Evaluation Map

	Code hardening	Guardrail optimization
Persistent artifact	Source tree and deployed service	External rules and guarded target
Accepted evidence	Reproduced exploit	Judge-accepted jailbreak
Blue update	Apply patch and restart service	Revise external rules
Security timing	Available vulnerability checks after update	Renewed attacks in the next Red round; retained replay separately
Security metric	TDSR: attack blocked and regression check passes	Current-round DSR: fraction of harmful targets without a successful attack; CRDE: blocking on a fixed retained attack set
Utility metric	Regression checks; SDR is the FDSR–TDSR gap	Three-prompt immediate screen; FPR on round-specific benign sets

Table 9: Domain-specific objects, timing, and metrics. The shared protocol does not imply numerical equivalence of DSR across domains.

D PROTOCOL AND EVIDENCE DETAILS

State, views, and memory boundary. The state decomposition separates three objects that iterative agent systems can otherwise conflate. The agent configuration—checkpoint, prompts, tools, and role assignment—defines the fixed policies. X_k is the mutable object under test: a source tree and deployed service in code hardening, or an external rule set and guarded target in content safety. C_k is an audit archive that associates accepted evidence, proposed changes, and the observations available for analysis. The archive need not be reproduced verbatim in either agent prompt. Instead, the task-specific views expose only the interfaces required by each role: Red probes the current artifact, Blue receives accepted attack evidence and edit access, and evaluators obtain the artifacts needed for checking or screening. Cross-round persistence is therefore supplied by X_{k+1}, not by assuming that an agent is given or remembers the full transcript, or that its parameters are implicitly updated.

State-indexed attacks, updates, and observations define the protocol's intended temporal provenance. An attack accepted on X_k conditions Δ_k, whose application produces X_{k+1} for subsequent security or utility checks. The available aggregate guard records preserve only partially aligned cross-table labels, so they support trajectory-level rather than complete event-level alignment. The abstraction does not treat the artifact as a lossless encoding of history or claim exhaustive repair.

Three protocol commitments. First, only attacks accepted by V_{atk} enter E_k and condition the Blue update. Second, the updated artifact is persistent: the next Red call probes the result of the previous Blue call. Third, the workflow records both a security observation and an available utility observation instead of treating attack failure alone as a successful repair. In code hardening, these elements are the source tree and deployed service, reproduced exploits, post-update vulnerability checks, and regression checks; in guardrail optimization, they are the external rule set, judge-accepted jailbreaks, renewed probing, and benign-input screening. The commitments define the feedback structure without imposing a common validator, payoff, or rollback rule.

Round execution and stopping. The Red Team probes S_k, the task-specific checker or judge retains accepted attack evidence E_k, and the Blue Team proposes Δ_k. Code updates are applied to the Docker service and subjected to the available post-update vulnerability and regression checks, with refinement allowed within the retry budget. Guardrail updates receive a benign-input screen and face renewed attack in the next Red round; retained attacks supply later diagnostics. The code workflow stops on null outputs, batch failure, a patience condition, or the five-epoch budget, whereas the guardrail study follows its fixed recorded trajectory. Because updates can leave unrelated weaknesses or expose new paths, each round remains an empirical test rather than a guaranteed improvement.

Trace diagnostics. Together, the reported utility trajectory, the OOP CMS regression, and the Qwen plateau show why an endpoint score is incomplete. OOP CMS is the second code environment under the recorded default backbone. Qwen3-Max is a separate backbone ablation, but the available experimental record does not identify which application it used; we therefore do not present it as two-environment evidence under Qwen3-Max. The supplement retains the two round-wise trajectories and aggregate token accounting.

Record-to-claim boundary. The reported example localizes unsanitized identifier interpolation in an SQL query in php_action/removeOrder.php and links the reproduced request to the stated patch-and-check workflow. A complete round record would associate the accepted reproduction on S_k with the affected file and root cause, the applied diff producing candidate S_{k+1}, restart status, and the available vulnerability and regression outcomes. This mapping explains how a transition would be audited; the manuscript does not contain per-attack records that confirm complete field coverage.

Requirements for a matched code comparison. A matched comparison requires the same initial repository and deployment, vulnerability inventory, model revisions, tools, budgets, and service checks. It must also use a common estimator: the cooperative cumulative union is not the final-state outcome reported for RvB. Replaying a fixed exploit set on each final artifact would distinguish retained protection from newly discovered attack coverage, while regression checks measure service preservation.