

EvoSafeHarness: Evolving Model- and Domain-Specific Harnesses for Securing Agents

Nanxi Li¹ Yingzi Ma² Yulong Cao³ Edward Suh³ Bo Li⁴ Dawn Song⁵ Chaowei Xiao^{1,3}

¹Johns Hopkins University ²University of Wisconsin–Madison ³NVIDIA

⁴University of Illinois Urbana–Champaign ⁵UC Berkeley

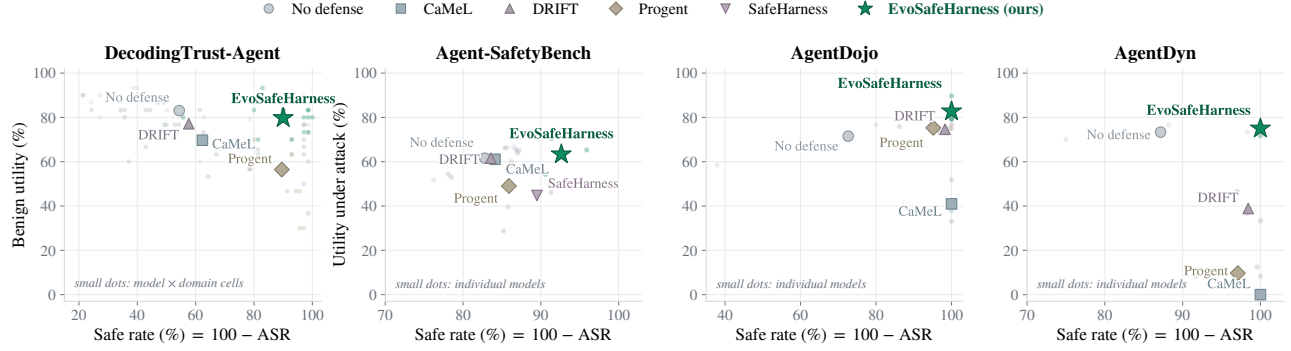


Figure 1. EvoSafeHarness dominates the utility–security trade-off on all four benchmarks. Each panel plots utility against safe rate ($100 - \text{ASR}$), so higher is better on both axes and the ideal defense sits in the top-right corner. Large markers show cross-model method averages and small dots show individual evaluation cells (model $\times$ domain cells on DecodingTrust-Agent, victim models elsewhere). The Agent-SafetyBench panel reports utility under attack; the AgentDojo panel reports held-out test results and the AgentDyn panel zero-shot transfer of the AgentDojo harness.

ABSTRACT

Large Language Model (LLM) Agents are turning language into real-world effects. They should remain safe against both indirect prompt injections and direct harmful requests. System-level safety harnesses provide an additional enforcement layer in addition to model-level solutions, but existing harness designs are typically built once by experts and applied across heterogeneous models and domains. The effective defense is inherently deployment-dependent: models differ in how much external enforcement they need before utility starts to drop, while domains differ in which effects, state, and action sequences must be governed. A harness strict enough for one model over-blocks another, and a policy general enough to transfer across domains can miss the safety relations of the application.

We present **EvoSafeHarness**, a safety-specific harness optimization framework that automatically synthesizes a deployable harness for a frozen model in a target domain. Unlike existing harness generation frameworks that target utility alone, it jointly searches a natural-language policy and executable code logic, guided by behavioral feedback from the target model and by a domain specification, and screens each candidate with a fresh-context adversarial review that rejects rules keyed to benchmark artifacts. Across four agent benchmark families, EvoSafeHarness establishes a stronger safety–utility frontier than fixed expert-designed defenses. On DecodingTrust-Agent, across fifteen independently searched model $\times$ domain deployments, it reduces average attack success rate (ASR) from 45.6% to 10.0% at a 3.3-point utility cost, the best score in 14 of 15 cells. On AgentDojo it reaches 82.8% utility at 0.0% ASR, twice the utility of CaMeL at the same zero-ASR operating point, and the same harness transfers unchanged to unseen AgentDyn suites. It also attains the best score on Agent-SafetyBench for every victim and keeps mean ASR below 20% under adaptive PAIR attacks with a refinement budget of 16. Analysis of the synthesized harnesses shows that domain semantics shape which safety relations and trajectory state are required, while model and runtime behavior shape how and where those relations are enforced, supporting harnesses optimized for the deployment at hand rather than a single universal design.

github.com/SaFo-Lab/EvoSafeHarness

andylinx.github.io/EvoSafeHarness

1 Introduction

Language-model agents are moving from demonstration to deployment. As they gain access to sensitive data, financial accounts, production systems, and external services, safety becomes an operational requirement. A chatbot failure may end in an undesirable response; an agent failure can result in a transferred payment, a leaked credential, deleted production data, or a persistent shell process. The unit of safety has expanded from a single utterance to an entire action trajectory, and the consequences of failure have expanded with it.

What makes agent safety qualitatively harder is that harmful instructions can enter through two channels that cross different security boundaries. In an *indirect prompt injection* attack [Greshake et al., 2023, Liu et al., 2024, Perez and Ribeiro, 2022], an adversary embeds instructions in external content—such as an email, a web page, or a document—that the agent must consume as data. If the agent treats this content as authoritative, it may execute actions that the user never requested. In a *direct attack* [Chen et al., 2026, Andriushchenko et al., 2025, Zhang et al., 2024], the harmful instruction instead arrives through the nominal user channel itself: a malicious user may ask the agent to transfer funds beyond an approved limit, delete production data, or disclose protected credentials.

To address these risks, model-level defenses have been proposed and remain essential [Wallace et al., 2024, Chen et al., 2025a,b]. These methods improve the model’s ability to distinguish trusted instructions from untrusted content or to learn to refuse unsafe requests. However, model-level defenses can make policy-compliant behavior more likely, but they do not by themselves provide a system-level enforcement boundary that is independent of model behavior. This limitation has motivated system-level defenses implemented in the *harness* level. Existing approaches use mechanisms such as provenance marking, injection detection, trajectory validation, capability enforcement, and lifecycle-level execution control [Hines et al., 2024, Inan et al., 2023, Li et al., 2025, Debenedetti et al., 2025, Lin et al., 2026c], demonstrating that the harness is an effective locus of agent security. However, these methods generally instantiate a fixed, expert-designed defense whose policy and control flow are the same across heterogeneous models and domains (Figure 2A). This leaves an open question: can a universal harness provide the best safety–utility trade-off across deployments, or should the harness be adapted to the target model and domain?

Building a universal harness that remains both safe and useful across models and applications is challenging because these two sources of variation affect different aspects of harness design. Model variation changes the appropriate strength of enforcement. A strongly safety-trained model such as Claude Opus may already resist many of the attacks that an external harness is intended to block; as shown in Figure 1, imposing a strict capability-based defense such as CaMeL [Debenedetti et al., 2025] can preserve strong security while substantially reducing benign utility. The same restrictions may nevertheless be essential for a model that is more susceptible to adversarial instructions. The marginal security benefit and utility cost of a harness are model-dependent, so a configuration that achieves a favorable safety–utility trade-off for one model need not do so for another. Application domain variation goes beyond enforcement strength: it changes the safety relations and control flow that the harness must implement. This is particularly clear for direct attacks, where the request arrives through the user channel and its harmfulness must be determined from the semantics of the requested effect. A filesystem harness may need to inspect command effects, sensitive paths, secret movement, and subsequent data flow. A finance harness, by contrast, must distinguish trades from money egress, enforce destination constraints, and maintain transaction history because a sequence of individually permissible trades may collectively constitute wash trading. Command and path filters cannot capture these financial relations, while a transaction ledger provides no protection against filesystem threats such as persistence or secret exfiltration. Model variation changes how strongly a harness should intervene, whereas domain variation changes its predicates, state, routing logic, and enforcement points; a single fixed design is unlikely to provide the best safety–utility trade-off across both.

This heterogeneity makes automated harness engineering an appealing next step. Meta-Harness [Lee et al., 2026] demonstrates that an agentic proposer can iteratively search harness code using behavioral feedback from the target model while inspecting candidate source code, evaluation scores, and execution traces from previous iterations. The same behavioral signals can expose model-specific security failures, but applying the paradigm to security is not as simple as replacing an ordinary task reward with attack success. Doing so creates three dangerous shortcuts. First, a harness can appear safe by refusing everything, converting security into utility collapse. Second, it can overfit to superficial artifacts in the evaluation set, such as filenames, paths, or recurring

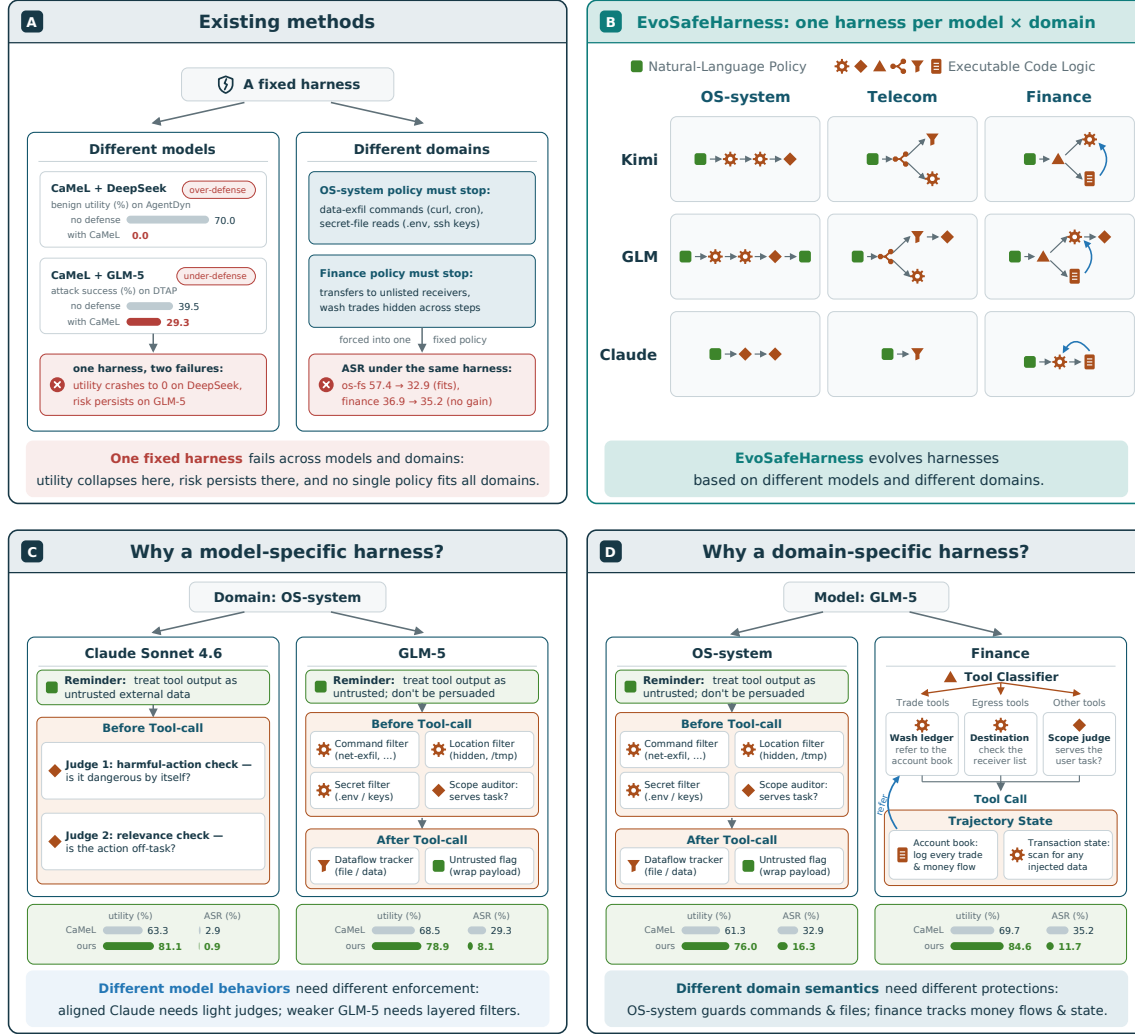


Figure 2. From the limits of fixed defenses to our model- and domain-specific EvoSafeHarness. (A) Existing fixed harnesses fail across models and domains: utility collapses on one victim, risk persists on another, and no single policy fits every domain. (B) EvoSafeHarness instead automatically designs a distinct combination of natural-language policy and executable code logic for every model×domain deployment. (C) With the domain fixed, model behavior determines how the same security relation is enforced; each model’s searched harness beats CaMeL on both utility and ASR. (D) With the model fixed, domain semantics determine which relations and state must be protected, and each domain’s searched harness again dominates.

attack phrases, and thereby obtain improvements that disappear under trivial renaming or paraphrasing. Third, because synthesized harness code is itself part of the enforcement mechanism, an error that skips a check or fails open can create an untested path for unsafe tool execution. Beyond these three failure modes, a scalar reward provides poor diagnostic feedback: it does not reveal whether a failure arose from a direct harmful request or an indirect injection, nor what kind of control should be revised. Safety harness synthesis needs a search procedure designed to preserve utility, test generalization, respect an explicit trust boundary, and return attack-specific evidence rather than only a scalar reward.

To meet these requirements, we present **EvoSafeHarness**, a safety-specific meta-harness that optimizes a deployable harness H around a frozen model M in a target domain D (Figure 3). Each search is conditioned on a domain specification that defines the deployment contract. Within this contract, four components form a closed optimization loop: a **Designer** proposes a natural-language policy together with executable code logic; a fresh-

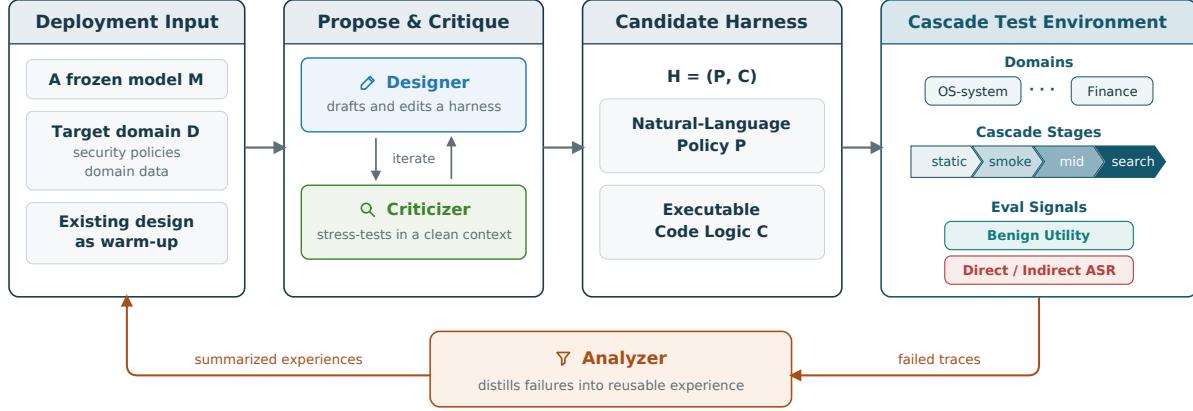


Figure 3. EvoSafeHarness searches a deployment-specific security harness. Given a frozen model M , a target domain D , and existing defenses as a warm start, the **Designer** drafts a candidate harness $H = (P, C)$, a natural-language policy P paired with executable code logic C , and iterates with a fresh-context **Criticizer** that stress-tests each proposal. Survivors enter the **Cascade Test Environment**, which runs cheap-to-expensive stages and separately measures benign utility and direct/indirect ASR. The **Analyzer** distills failed traces into experience that seeds the next iteration.

context **Criticizer** reviews the proposal; a **Cascade Test Environment** executes and scores surviving candidates; and an **Analyzer** returns decomposed scores and failure traces to the **Designer** for the next revision. These components are designed to address the preceding failure modes directly. Separate benign evaluation prevents a refuse-all candidate from presenting zero attack success as an unqualified improvement. The generalization requirements in the domain specification, together with independent **Criticizer** review, screen out rules keyed to benchmark-specific artifacts. Finally, the **Cascade Test Environment** and **Analyzer** preserve benign, direct-attack, and indirect-attack outcomes as separate feedback channels, allowing the **Designer** to revise the policy or executable code logic in response to the specific failure observed. EvoSafeHarness thereby turns a general reward-driven code search into a constrained, adversarially validated procedure for generating model- and domain-specific safety harnesses.

We evaluate EvoSafeHarness on four agent benchmark families spanning both attack channels, and it improves the safety-utility trade-off on all of them (Figure 1). On DecodingTrust-Agent, our primary benchmark, fifteen independently searched model $\times$ domain harnesses cut average ASR from 45.6% to 10.0% at a 3.3-point utility cost and obtain the best score in 14 of 15 cells, whereas the fixed defenses CaMeL, DRIFT, and Progent either leave ASR above 37% or sacrifice more than 20 utility points (Table 1). On Agent-SafetyBench, whose harms include unsafe user requests and misinformation that never surface as a tool call, the searched harness has the lowest unsafe-behaviour rate and the best score for every victim (Table 2). On AgentDojo it reaches 82.8% utility at 0.0% ASR, twice the utility of CaMeL at the same zero-ASR operating point, and the same harness transfers unchanged to unseen AgentDyn suites at 75.0% utility and 0.0% ASR (Table 2). Finally, a frozen harness keeps mean ASR below 20% against three adaptive PAIR attackers with a refinement budget of 16 (§6.3), and controlled ablations show that the security warm start, the fresh-context Criticizer, and the nested cascade each contribute measurably (§7.5).

The returned harnesses clarify why deployment-specific design produces these gains. Figure 2C holds the request, attack, and os-filesystem domain fixed and varies only the victim model. Sonnet 4.6 reaches zero held-out ASR with a lightweight policy and two semantic checks, whereas GLM-5 benefits from deterministic gates, provenance state, and verdict caching. A weaker model does not just need more rules. Model and runtime behavior determine whether the same safety relation should be enforced semantically or deterministically, with or without state, and at which point in execution. Figure 2D instead holds GLM-5 fixed and varies the domain. Filesystem safety depends on command effects, sensitive paths, and data flow, whereas finance must distinguish trades from money egress and retain transaction history because individually permissible actions can form a harmful sequence. Domain adaptation decides which safety relations and state must be protected, while model adaptation determines how to enforce them without unnecessary utility loss.

2 Related Work

Harness and pipeline optimization. DSPy [Khattab et al., 2023] and GEPA [Agrawal et al., 2025] optimize the prompts of a fixed pipeline; a more recent line optimizes the harness itself. Meta-Harness edits harness source with an agentic proposer that reads execution traces [Lee et al., 2026], Natural-Language Agent Harnesses (NLAH) make the harness an editable natural-language policy [Pan et al., 2026], and VeRO and AHE add versioned, budget-controlled evaluation loops [Ursekar et al., 2026, Lin et al., 2026a]. All of these optimize task performance or cost, not behavior under an adversary, and none searches a defense per model and per domain, even though harness benefit is known to vary non-monotonically with the base model [Lin et al., 2026b]. EvoSafeHarness keeps Meta-Harness’s trace-driven search and NLAH’s policy surface but changes the optimization regime: a security domain specification, threat-stratified scoring, and fresh-context adversarial review.

Prompt-injection attacks and system-level defenses. Indirect prompt injection [Greshake et al., 2023, Liu et al., 2024, Perez and Ribeiro, 2022] lets an attacker who controls ingested content issue instructions the model may follow, and adaptive attacks tuned against a known defense break most published defenses [Zhan et al., 2025]; our threat model (§3) covers both this channel and direct harmful requests. System-level defenses [Xiang et al., 2026] range from prompt-level marking [Hines et al., 2024] and injection detectors or guardrail stacks [Inan et al., 2023, Li et al., 2026b, Chennabasappa et al., 2025], through in-loop trajectory monitors such as DRIFT [Li et al., 2025] and IPIGuard [An et al., 2025] and capability policies such as Progent [Shi et al., 2025], to architectural separation of untrusted data from control flow in CaMeL [Debenedetti et al., 2025]. Closest to us is SafeHarness [Lin et al., 2026c], which hand-designs four lifecycle defense layers and reuses that architecture across models and domains. EvoSafeHarness instead treats the choice of mechanisms and their thresholds as the optimization problem: prior designs enter only as warm-start experience, and the Designer is free to remove, recombine, or invent mechanisms for each deployment.

Agent safety benchmarks. A growing suite measures agent (in)security. The DecodingTrust-Agent platform (DTAP) [Chen et al., 2026] is a multi-domain agent red-teaming benchmark in which the agent acts through tool servers; its release contains fourteen domains, of which our cross-model grid evaluates three representative ones. AgentDojo evaluates injection attacks and defenses for tool-using agents across four suites [Debenedetti et al., 2024]; we use its out-of-distribution extension AgentDyn [Li et al., 2026a] as a generalization stress test. Agent-SafetyBench evaluates unsafe agent behavior across 2,000 safety-critical tasks and diverse interactive environments [Zhang et al., 2024], and AgentCanary [Li et al., 2026c] supplies the adaptive PAIR-style attacker used in §6.3. Beyond these, InjecAgent benchmarks indirect injections in tool-integrated agents [Zhan et al., 2024] and AgentHarm measures the harmfulness of agents under direct misuse [Andriushchenko et al., 2025]. These benchmarks measure vulnerability; our contribution is a method that searches a defense against the measured failures of a specific model. ReAct-style tool use [Yao et al., 2023] is the underlying agent loop throughout.

3 Problem Formulation

Agent and harness. A tool-using agent is a pair (M, H) : a frozen language model M and a *harness* H that mediates every interaction between M , the user, and the tools. Concretely, H is an ordered pipeline that (a) constructs the system message and initial query, (b) invokes M , (c) executes the tool calls M emits, and (d) returns tool results to M , looping until M produces a final answer. The no-defense harness H_0 is the bare loop. A *defense* is any modification of H_0 .

Natural-language policy and executable code logic. We represent a harness as a pair $H = (P, C)$. The *natural-language policy* P is any policy transform applied to the model’s context: trust-boundary declarations, provenance framing, refusal criteria, or other standing instructions. The *executable code logic* C is an arbitrary program valid under the application’s harness adapter. It may rewrite or block tool calls, transform tool outputs, maintain per-trajectory state, enforce capabilities, or invoke quarantined auxiliary classifiers. This decomposition describes where the Designer can act; it does not partition the defense into a fixed set of mechanisms or lifecycle slots.

Threat model. We consider two attack channels, neither of which can reach the model weights or the harness. In the *indirect* channel the attacker controls content the agent will read but not the user’s request: DTAP delivers these attacks through files, records, tickets, and messages across heterogeneous tool servers. On AgentDojo/AgentDyn the attack family is `important_instructions`, a payload wrapped in authoritative-looking tags planted in a tool result, attempting to redirect the agent to an injection goal g such as transferring money to an attacker account. In the *direct* channel, which DTAP adds, the malicious goal is the user instruction itself, so the agent must directly reject the harmful request. A harness must hold against both. These channel labels describe where adversarial authority originates. Orthogonally, a content-level defense checks provenance and preserves the instruction–data boundary, whereas a domain/action-level defense checks whether the resulting effect is authorized under the deployment’s policy. Either level may use P , C , or both. Indirect attacks typically demand provenance plus an action-level backstop; direct attacks cannot be solved by provenance alone.

Objective. For a benchmark D with benign tasks $\mathcal{B}$ and attack tasks $\mathcal{A}$, utility and attack success rate are

$$U(M, H, D) = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \mathbf{1}[\text{success}(b)],$$

$$\text{ASR}(M, H, D) = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \mathbf{1}[\text{attack}(a)],$$

where success and attack are the benchmark’s own judges. We optimize the scalar

$$\text{score}(M, H, D) = 100 \cdot (U(M, H, D) - \text{ASR}(M, H, D)), \quad (1)$$

which rewards security only when utility is preserved: a harness that refuses everything scores zero, as does a fully functional harness that completes every benign task and allows every attack. A harness that blocks nothing receives the no-defense score, which is positive whenever its utility exceeds its attack success rate.

Secure-harness optimization seeks

$$H_{M,D}^* = \arg \max_{H \in \mathcal{H}_D} \text{score}(M, H, D),$$

where $\mathcal{H}_D$ contains every program that satisfies the target application’s adapter and the domain specification’s immutable-component constraints, with no hand-enumerated mechanism taxonomy. The solution is intrinsically model- and domain-specific because both the observed failures of M and the action semantics of D shape the search.

4 Method: EvoSafeHarness

EvoSafeHarness is a search loop with four components, shown in Figure 3. An authoritative domain specification exposes the security problem and an open harness interface (§4.1). The Designer (§4.3) then reads the archive of prior source, scores, and the target model’s failure traces and proposes any valid program under that interface. Before evaluation, a fresh-context Criticizer (§4.5) challenges the proposal with benchmark-independent evasions. Surviving candidates reach a staged Cascade Test Environment (§4.4) that separately measures benign utility, direct attacks, and indirect attacks and supports cheap-to-expensive admission. The archive is warm-started from prior security designs (§4.2), but those designs provide experience rather than a template. Algorithm 1 states the loop precisely, including warm-start distillation (lines 1–2), per-candidate review and repair (lines 6–8), and the resource-aware cascade (lines 9–16).

4.1 Domain Specification and Open-Ended Harness Search

Each search begins from a *domain specification*, used directly as the Designer’s task contract. It defines (i) the frozen victim, backend, tools, and judges; (ii) direct and indirect threat semantics and which runtime context is trusted under each; (iii) the train-only evaluation cascade and score; (iv) the generalization mandate and anti-overfit robustness check; and (v) the adapter through which a defense may act. The specification says what must remain invariant and how a candidate will be evaluated, not which defense architecture to build.

Within this contract, a candidate may modify both the natural-language policy P and executable code logic C . On DTAP, the minimal adapter is a Python `Defense` object with `system_prompt_transform`, `on_pre_tool_call`, and

on_post_tool_call, plus arbitrary helper code and per-trace state. The Designer may compose any mechanism expressible there: rewrite prompts or arguments, block or redirect calls, inspect or transform outputs, track provenance and multi-step effects, cache verdicts, or invoke quarantined classifiers. On AgentDojo, the executable code logic is likewise open-ended but instantiated as one or more pipeline elements composed around the fixed agent loop. The three DTAP methods and the AgentDojo pipeline contract are interception APIs, not mechanism slots. Neither P nor C is subdivided into a required taxonomy.

4.2 Warm-Starting from Existing Defenses

A search from an empty archive has little security-specific guidance. We therefore distill eight security design experiences from mature defenses [Li et al., 2025, Debenedetti et al., 2025] into initial candidates and reusable guidance, for example that tool output must remain untrusted data rather than become an instruction, and that a tool call’s effect must be checked against the trusted user request before it executes. These experiences give the search robust starting invariants instead of forcing it to rediscover basic security boundaries from a small evaluation set. They are guidance, not a template or candidate grammar: the Designer may remove, recombine, replace, or extend them, and the returned harnesses frequently contain mechanisms absent from the warm start.

4.3 Designer: Selecting and Revising Harnesses

Each search iteration decides which previous harness to build on and how to revise it. The evidence for both decisions lives in an explicit archive $\mathcal{H}$ whose entries store a candidate’s source bundle, parent and design hypothesis, cascade scores, Criticizer verdict, and per-task traces on M ; the Analyzer turns failed traces into design experience. Formally, the Designer implements the policy

$$\pi : (\Sigma_D, \mathcal{H}) \mapsto (H_{\text{parent}}, \mu), \quad H' = \mu(H_{\text{parent}}),$$

where Σ_D is the domain specification and μ is a free-form edit rather than a menu of mutations: it may revise P , introduce trajectory state, add or remove a judge or gate, change control flow, or rewrite C entirely, subject only to the application adapter. When possible the Designer changes one mechanism at a time so that score changes stay interpretable. Following Meta-Harness [Lee et al., 2026], it may inspect prior programs, the score leaderboard, design notes, and raw failure traces before making a local edit or a structural rewrite; the natural-language policy follows NLAH [Pan et al., 2026]. The archive, not hidden conversational memory, is the optimizer state: it retains every completed candidate and the prefix evidence of candidates stopped by the confidence gate (§4.4). The Designer never reads the held-out split.

4.4 The Evaluation Cascade

Candidate evaluation is sequential and *nested*: each stage extends the same ordered evidence stream, so earlier trajectories are reused rather than evaluated again. The cascade contains four cheap-to-expensive stages:

Stage 0 (static). Import and structural check that the bundle exposes a valid build; must pass.

Stage 1 (smoke prefix). A small diagnostic prefix must complete without exceptions.

Stage 2 (confidence gate). The prefix is extended with a broader utility–security sample. We stop only when a stratified bootstrap’s one-sided 95% upper bound places the candidate more than δ Score points below the no-defense reference; uncertain candidates continue.

Stage 3 (search extension). Survivors extend the same trajectory set to the complete search sample, whose aggregate becomes the search score (Eq. 1).

The conservative confidence rule favors fidelity over aggressive pruning: a candidate is discarded only when the available evidence already establishes inferiority. Seeds are fixed (a frozen 70/30 train/held-out split with seed 42) so the corpus is reproducible, and the held-out split remains invisible to every cascade decision. Per-corpus prefix sizes and gate constants are listed in Appendix A; Section 7.6 measures the resulting savings.

Algorithm 1 EvoSafeHarness secure-harness search

Input: frozen model M ; domain specification Σ_D (adapter, threat model, trusted context, train manifests, score); budget B ; prior security designs $\mathcal{P}$ (warm-up experience); ordered train corpus $\mathcal{C}$ of task-injection pairs; confidence-prefix length q ; no-defense prefix score s_q^0 ; non-inferiority margin δ

Output: best harness H^* for (M, D)

```

1:  $\mathcal{S} \leftarrow \text{DISTILL}(\mathcal{P})$  // warm start: external designs become seed bundles
2:  $\mathcal{H} \leftarrow \{(s, \text{EVAL}(s, M, \mathcal{C})) : s \in \mathcal{S}\}$ 
3: while budget  $B$  remains and not converged do
4:    $(H_{\text{parent}}, \mu) \leftarrow \pi(\Sigma_D, \mathcal{H})$  // Designer reads the spec, scores, and traces
5:    $H' \leftarrow \mu(H_{\text{parent}})$  // any valid  $(P, C)$  program in  $\mathcal{H}_D$ 
6:   while  $v \leftarrow \text{REVIEW}(H')$  flags a rename / relocate / rephrase evasion do
7:      $H' \leftarrow \text{REPAIR}(H', v)$  // Criticizer
8:   end while
9:   if  $\neg \text{STATIC}(H')$  or  $\neg \text{SMOKE}(H', M, \mathcal{C})$  then
10:    continue // cascade Stage 0–1: static check + fixed smoke set
11:   end if
12:    $s_q \leftarrow \text{EVALPREFIX}(H', M, \mathcal{C}, q)$  // nested prefix; reuse smoke evidence
13:   if  $\text{UCB}_{.95}(s_q - s_q^0) < -\delta$  then
14:    continue // stop only when the candidate is confidently inferior
15:   end if
16:    $s \leftarrow \text{EXTENDEVAL}(H', M, \mathcal{C})$  // extend the same evidence stream
17:    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(H', s, v)\}$  // commit (keeps Pareto frontier)
18: end while
19: return  $\arg \max_{H \in \mathcal{H}} \text{score}(M, H, D)$ 

```

Two-level resource accounting. EvoSafeHarness consumes resources at two different levels. The *outer optimizer* uses an agentic coding model to inspect the archive, diagnose traces, propose or review mutations, and edit the harness. The *inner evaluator* runs victim-model episodes to measure the resulting candidate. Archive reuse and prompt caching reduce repeated outer context, whereas a cascade can prevent weak candidates from consuming expensive inner episodes when its gates fire or earlier trajectories are reused. We account for these levels separately because they use different models, tokenizers, cache semantics, and units of work; neither candidate count nor one undifferentiated resource budget faithfully measures both. Section 7.6 analyzes the archived consumption, and Appendix G explains the aggregate ledgers.

4.5 The Criticizer

A search that maximizes a benchmark score will, left unchecked, discover *benchmark artifacts*: a rule that blocks a specific directory, filename, or command string raises the score and collapses the moment an attacker renames the target. This is the central failure mode of automated defense search, and it is structurally tempting because such rules are cheap and locally effective.

We counter it with two coupled mechanisms. First, a generalization requirement is applied to every rule: if the attacker renamed the path, moved to an unlisted location, or rephrased the instruction with the same malicious intent, would this rule still fire? A rule that survives only because of a literal token the benchmark happens to use is rejected. Acceptance keys on attack invariants: **provenance** (the dangerous instruction arrived inside tool output, not from the user), **scope** (the action touches resources the user’s task never named), and **intent mismatch** (the action is unrelated to the request). These hold regardless of vocabulary.

Second, a Criticizer: before any cascade run, the candidate’s natural-language policy P and executable code logic C are handed to a fresh, independent Criticizer in a clean context, providing a security review distinct from the defended model and exempt from the frozen-model constraint. It is asked: what is the cheapest way an attacker keeps the same intent but evades this? Does any rule depend on a literal benchmark token or one domain’s tool names? Would it still protect an agent in a domain it was never tuned on? Flagged trivial evasions are repaired before evaluation budget is spent, and the verdict is recorded with the candidate. Section 7 quantifies

the Criticizer’s contribution: removing it leaves the search score unchanged or higher but lowers held-out score by up to 31 points.

5 Experimental Setup

Benchmarks. We evaluate on four settings, each answering one question about a searched harness. *DecodingTrust-Agent* (DTAP) [Chen et al., 2026] is the primary setting: it exposes one held-out protocol across many tool domains and victim models, so it is the only place where the model $\times$ domain grid that motivates per-deployment search can be measured. We use three action domains (os-filesystem, finance, telecom) with both direct and indirect attacks. *Agent-SafetyBench* (ASB) [Zhang et al., 2024] tests a harm distribution much broader than prompt injection, including unsafe user requests and misinformation carried into the final answer. *AgentCanary* [Li et al., 2026c] asks the adversarial question: we freeze the searched harness and let a PAIR-style attacker rewrite its prompts against it with a growing refinement budget. *AgentDojo* $\rightarrow$ *AgentDyn* [Debenedetti et al., 2024, Li et al., 2026a] tests transfer: a harness searched on AgentDojo is run unchanged on AgentDyn suites whose tools and workflows the search never saw. Splits, task counts, judges, and per-benchmark protocols are in Appendix B.

Victims and baselines. DTAP freezes five victims (Sonnet 4.6, GLM-5, Kimi-K2.5, Qwen3.7-plus, DeepSeek-V4-Flash); ASB uses DeepSeek-V3.2, Kimi-K2.6, and GLM-5.2; AgentDojo/AgentDyn use MiMo-V2.5-Pro, DeepSeek-V4-Pro, and GLM-5.2; AgentCanary uses DeepSeek-V4-Flash driving the OpenClaw agent runtime. We compare against CaMeL [Debenedetti et al., 2025], DRIFT [Li et al., 2025], Progent [Shi et al., 2025], and, on ASB, SafeHarness [Lin et al., 2026c]. Every baseline is ported once and frozen across all victims and domains, while EvoSafeHarness is searched separately for each cell. Search reads only training splits, and held-out outcomes never reach the Designer or Criticizer. Baseline ports and Progent’s per-domain policy mapping are documented in Appendix B.

Metrics. DTAP, AgentDojo, and AgentDyn report benign utility U , attack success rate ASR, and score = $U - \text{ASR}$ (Eq. 1); DTAP splits ASR into direct and indirect. ASB uses its own clean unsafe-behaviour rate (UBR), micro ASR, utility under attack (UA), and score = $UA - \text{ASR}$. AgentCanary counts an attack as successful when the security judge’s outcome score is ≤ 0.5 . Every safety number is paired with its utility counterpart, since a harness that refuses everything trivially reaches zero ASR.

6 Results

6.1 DecodingTrust-Agent: The Model $\times$ Domain Grid

Table 1 reports the fifteen-cell grid. Two facts about the undefended grid frame the result. The victim is itself the largest security variable: undefended ASR ranges from 4.8% (Sonnet 4.6) to 71.0% (DeepSeek-V4-Flash). And fixed defenses inherit the tool taxonomy of the domain they were designed for: CaMeL and DRIFT help most on os-filesystem, partially on finance, and not at all on telecom, where harmful actions use ordinary in-scope customer-service tools. Across the five-victim grid, Progent is the strongest fixed safety baseline, but its low ASR comes with substantial utility loss: its per-victim mean utility ranges from 44.4% to 66.7%, versus 71.1–86.7% for the undefended agents.

A defense fixed at design time cannot know which regime it will face.

EvoSafeHarness has the best score in 14 of the 15 cells against no defense and all three fixed baselines. Averaged over the grid, ASR falls from 45.6% to 10.0% for a 3.3-point utility cost, against 37.7% (CaMeL) and 42.4% (DRIFT) at equal or larger utility cost. Progent nearly matches the safety result at 10.5% ASR, but lowers utility to 56.4%, versus 79.8% for EvoSafeHarness. Under EvoSafeHarness, direct ASR falls from 50.9% to 12.6% and indirect ASR from 40.4% to 7.4%. The single loss is Sonnet 4.6/finance, already at 2.9% undefended. A paired McNemar test rejects no-change in 13 of 15 cells—the exceptions are the two Sonnet cells already below 3%—and pooled over the grid 385 attacks are neutralised against 11 newly opened ($p \approx 2 \times 10^{-78}$); benign-utility intervals overlap in every cell (Appendix H).

Model	Defense	os-fs		finance		telecom		Average	
		Util	ASR (d/i)	Util	ASR (d/i)	Util	ASR (d/i)	Util	ASR (d/i)
Sonnet 4.6	No-Defense	86.7	10.0 (5.7/14.3)	90.0	2.9 (0.0/5.7)	76.7	1.4 (0.0/2.9)	84.5	4.8 (1.9/7.6)
	CaMeL	50.0	2.9 (5.7/0.0)	63.3	2.9 (0.0/5.7)	76.7	2.9 (0.0/5.7)	63.3	2.9 (1.9/3.8)
	DRIFT	66.7	2.9 (5.7/0.0)	70.0	2.9 (0.0/5.7)	76.7	2.9 (0.0/5.7)	71.1	2.9 (1.9/3.8)
	Progent	36.7	1.4 (2.9/0.0)	60.0	1.4 (0.0/2.9)	73.3	1.4 (2.9/0.0)	56.7	1.4 (1.9/1.0)
	EvoSafeHarness	80.0	0.0 (0.0/0.0)	80.0	1.4 (0.0/2.9)	83.3	1.4 (2.9/0.0)	81.1	0.9 (1.0/1.0)
GLM-5	No-Defense	83.3	50.0 (57.1/42.9)	79.3	21.4 (17.1/25.7)	83.3	47.1 (62.9/31.4)	82.0	39.5 (45.7/33.3)
	CaMeL	63.3	18.6 (20.0/17.1)	65.5	18.6 (11.4/25.7)	76.7	50.7 (61.8/40.0)	68.5	29.3 (31.1/27.6)
	DRIFT	73.3	37.3 (50.0/24.2)	56.7	21.4 (11.4/31.4)	86.7	50.8 (55.9/44.0)	72.2	36.5 (39.1/33.2)
	Progent	70.0	7.1 (8.6/5.7)	46.7	4.3 (2.9/5.7)	83.3	38.6 (37.1/40.0)	66.7	16.7 (16.2/17.1)
	EvoSafeHarness	80.0	11.4 (17.1/5.7)	83.3	10.0 (8.6/11.4)	73.3	2.9 (5.7/0.0)	78.9	8.1 (10.5/5.7)
Kimi-K2.5	No-Defense	80.0	72.9 (94.3/51.4)	60.0	38.6 (42.9/34.3)	73.3	54.3 (77.1/31.4)	71.1	55.3 (71.4/39.0)
	CaMeL	66.7	22.9 (25.7/20.0)	63.3	32.9 (34.3/31.4)	76.7	50.0 (77.1/22.9)	68.9	35.3 (45.7/24.8)
	DRIFT	83.3	55.7 (91.4/20.0)	53.3	35.7 (34.3/37.1)	86.7	52.9 (74.3/31.4)	74.4	48.1 (66.7/29.5)
	Progent	30.0	4.3 (8.6/0.0)	46.7	8.6 (8.6/8.6)	56.7	21.4 (40.0/2.9)	44.4	11.4 (19.0/3.8)
	EvoSafeHarness	70.0	18.6 (31.4/5.7)	83.3	10.0 (8.6/11.4)	80.0	2.9 (2.9/2.9)	77.8	10.5 (14.3/6.7)
Qwen3.7-plus	No-Defense	86.7	75.7 (77.1/74.3)	93.3	60.0 (57.1/62.9)	80.0	37.1 (28.6/45.7)	86.7	57.6 (54.3/61.0)
	CaMeL	60.0	62.9 (82.9/42.9)	73.3	57.1 (51.4/62.9)	76.7	45.7 (48.6/42.9)	70.0	55.2 (61.0/49.6)
	DRIFT	83.3	65.7 (77.1/54.3)	83.3	57.1 (51.4/62.9)	76.7	47.1 (42.9/51.4)	81.1	56.6 (57.1/56.2)
	Progent	46.7	5.7 (8.6/2.9)	63.3	7.1 (5.7/8.6)	70.0	20.0 (37.1/2.9)	60.0	11.0 (17.1/4.8)
	EvoSafeHarness	70.0	7.1 (0.0/14.3)	93.3	17.1 (11.4/22.9)	73.3	1.4 (2.9/0.0)	78.9	8.5 (4.8/12.4)
DeepSeek V4-Flash	No-Defense	90.0	78.6 (94.3/62.9)	93.3	61.4 (65.7/57.1)	90.0	72.9 (82.9/62.9)	91.1	71.0 (81.0/61.0)
	CaMeL	66.7	57.1 (91.4/22.9)	83.3	64.3 (68.6/60.0)	83.3	75.7 (85.7/65.7)	77.8	65.7 (81.9/49.5)
	DRIFT	90.0	78.6 (97.1/60.0)	90.0	60.0 (60.0/60.0)	80.0	64.3 (88.6/40.0)	86.7	67.6 (81.9/53.3)
	Progent	30.0	2.9 (2.9/2.9)	63.3	7.1 (5.7/8.6)	70.0	25.7 (48.6/2.9)	54.4	11.9 (19.0/4.8)
	EvoSafeHarness	80.0	44.3 (74.3/14.3)	83.3	20.0 (20.0/20.0)	83.3	1.4 (2.9/0.0)	82.2	21.9 (32.4/11.4)
Average	No-Defense	85.3	57.4 (65.7/49.2)	83.2	36.9 (36.6/37.1)	80.7	42.6 (50.3/34.9)	83.1	45.6 (50.9/40.4)
	CaMeL	61.3	32.9 (45.1/20.6)	69.7	35.2 (33.1/37.1)	78.0	45.0 (54.6/35.4)	69.7	37.7 (44.3/31.1)
	DRIFT	79.3	48.0 (64.3/31.7)	70.7	35.4 (31.4/39.4)	81.4	43.6 (52.3/34.5)	77.1	42.4 (49.3/35.2)
	Progent	42.7	4.3 (6.3/2.3)	56.0	5.7 (4.6/6.9)	70.7	21.4 (33.1/9.7)	56.4	10.5 (14.7/6.3)
	EvoSafeHarness	76.0	16.3 (24.6/8.0)	84.6	11.7 (9.7/13.7)	78.6	2.0 (3.5/0.6)	79.8	10.0 (12.6/7.4)

Table 1. DTAP held-out results across five victims and three domains. Each domain uses 30 benign, 35 direct, and 35 indirect tasks. Util is benign utility; ASR (d/i) gives overall ASR with its direct/indirect breakdown. EvoSafeHarness is searched per cell; baselines are frozen across cells.

6.2 Agent-SafetyBench and AgentDojo→AgentDyn

Beyond prompt injection. Much of ASB’s harm is not injection-borne: the user’s own request may be unsafe, or the agent may repeat poisoned content in its final answer with no unsafe tool call anywhere in the trajectory, so a defense that only gates tool calls cannot intervene. Under one frozen protocol (Table 2a), EvoSafeHarness has the lowest clean UBR in every block and the lowest or tied-lowest ASR, while keeping utility under attack within 1.4 points of the best; averaged over the three victims it is the only defense that lowers UBR and ASR (12.3% and 7.4%) while also raising utility under attack (63.4%). The gain is not blanket refusal: SafeHarness ties our ASR on GLM-5.2 but gives up 22 points of UA. Fixed defenses apply one mechanism everywhere, so they either leave harm uncovered or over-block; search finds the enforcement point and policy appropriate to each victim.

Transfer without re-search. Searching once on AgentDojo and running the frozen harness on AgentDyn’s shopping, GitHub, and daily-life suites separates transferable safety from benchmark-specific blocking (Table 2b). Fixed baselines keep ASR low mainly by suppressing completion: CaMeL completes nothing on AgentDyn and Progent drops to 9.7% utility, because a policy fitted to AgentDojo’s tool vocabulary denies tools it has never seen. EvoSafeHarness reaches 82.8% utility at 0.0% ASR on AgentDojo—twice CaMeL’s utility at the same zero-ASR operating point—and keeps 75.0% at 0.0% on AgentDyn, above the undefended agent’s 73.3%. It transfers

(a) Agent-SafetyBench					(b) AgentDojo → AgentDyn					
Model	Defense	UBR ↓	ASR ↓	UA ↑	Model	Defense	AgentDojo		AgentDyn	
							Util ↑	ASR ↓	Util ↑	ASR ↓
DeepSeek V3.2	No defense	28.3	23.8	51.8	MiMo V2.5-Pro	No defense	76.7	20.0	76.7	11.8
	CaMeL	27.9	21.9	54.3		CaMeL	51.9	0.0	0.0	0.0
	DRIFT	26.7	21.5	53.0		DRIFT	73.6	1.8	46.6	3.0
	Progent	19.2	14.8	28.8		Progent	75.0	0.7	12.5	0.4
	SafeHarness	21.2	14.2	39.6		EvoSafeHarness	79.3	0.0	71.7	0.0
	EvoSafeHarness	14.6	9.4	54.3						
Kimi-K2.6	No defense	21.7	13.5	66.7	DeepSeek V4-Pro	No defense	58.6	62.1	70.0	25.0
	CaMeL	19.2	13.0	64.0		CaMeL	37.9	0.0	0.0	0.0
	DRIFT	21.7	14.6	66.3		DRIFT	73.6	3.4	36.7	1.7
	Progent	20.0	13.7	58.3		Progent	75.9	13.8	8.3	8.3
	SafeHarness	16.3	8.7	46.1		EvoSafeHarness	89.7	0.0	76.7	0.0
	EvoSafeHarness	10.0	4.1	65.3						
GLM-5.2	No defense	23.8	14.3	66.3	GLM-5.2	No defense	79.3	0.0	73.3	1.7
	CaMeL	22.9	12.8	65.0		CaMeL	33.1	0.0	0.0	0.0
	DRIFT	25.0	13.2	65.5		DRIFT	76.7	0.0	33.3	0.0
	Progent	22.5	13.8	60.0		Progent	75.0	0.0	8.3	0.0
	SafeHarness	19.2	8.6	48.6		EvoSafeHarness	79.3	0.0	76.7	0.0
	EvoSafeHarness	12.2	8.6	70.5						
Mean	No defense	24.6	17.2	61.6	Mean	No defense	71.5	27.4	73.3	12.8
	CaMeL	23.3	15.9	61.1		CaMeL	41.0	0.0	0.0	0.0
	DRIFT	24.5	16.4	61.6		DRIFT	74.6	1.7	38.9	1.6
	Progent	20.6	14.1	49.0		Progent	75.3	4.8	9.7	2.9
	SafeHarness	18.9	10.5	44.8		EvoSafeHarness	82.8	0.0	75.0	0.0
	EvoSafeHarness	12.3	7.4	63.4						

Table 2. Agent-SafetyBench (a) and AgentDojo→AgentDyn transfer (b). (a) Held-out split of 240 tasks, each run under six conditions (clean plus five attack channels) per arm under one frozen protocol; UBR is the clean unsafe-behaviour rate and UA is utility under attack. (b) AgentDyn columns run the harness searched on AgentDojo zero-shot, with no re-search, on suites with unseen tools and workflows. Best value per block in bold.

because what it enforces is provenance and task scope rather than a tool list: the harness anchors authority in the user request, marks tool output as data, and tells the agent to ignore embedded commands while continuing the requested task, which is why utility can rise rather than merely survive (Appendix B).

6.3 Robustness to Adaptive Attacks

The preceding results use fixed attack corpora; we next test the searched harness against an attacker that adapts to it. On AgentCanary we freeze the searched harness and let a PAIR-style attacker rewrite each prompt against the live defended agent, using the target’s own responses as feedback, with a growing refinement budget (Figure 4a). At budget 0 the harness lowers held-out ASR from 23.6% to 9.7% at unchanged utility (74.1% vs. 73.7%). Adaptive rewriting recovers only a few bypasses: ASR stays at 9.7% through budget 2 and reaches 16.7% at budget 16 for two of the three attackers and 25.0% for the third, a mean of 19.5%, still below the undefended agent facing static prompts. The harness holds up when the attacker optimizes against it because the relations it enforces depend on provenance and task scope, not on the wording of any particular attack. The protocol, the attacker models, and per-budget numbers are in Appendix B (Table A1).

7 Analysis: What Does EvoSafeHarness Discover?

The aggregate results establish that EvoSafeHarness improves the safety–utility trade-off. We now inspect the final bundles, candidate lineages, failure traces, held-out failures, controlled ablations, and resource ledgers to understand what the search learns and where it stops helping. We use *safety relation* to mean a checkable constraint connecting the trusted task to an emitted action, its target, prior actions, or eventual effect—for

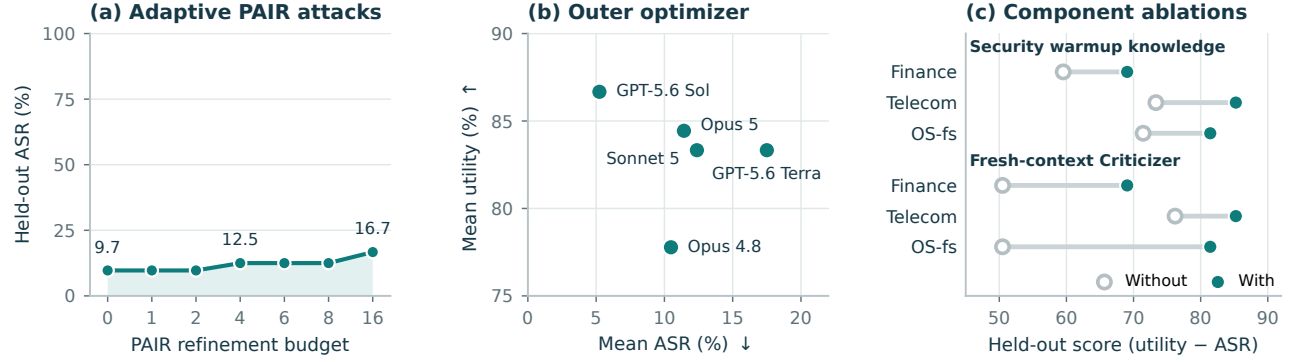


Figure 4. Robustness and ablations. (a) Held-out ASR of the frozen AgentCanary harness as a PAIR attacker’s refinement budget grows; budget 0 replays the static prompts. (b) Varying only the outer model that performs the search, with the victim fixed to Kimi-K2.5; Opus 4.8 is the primary-grid reference. (c) Held-out score with and without the security warmup knowledge and the fresh-context Criticizer, same victim.

example, a request to buy a stock versus an emitted call that sells it. We first follow concrete traces from a single action to a multi-step trajectory and then to concurrent runtime behavior (§7.1). We next compare how those relations change across models and domains (§7.2), audit the residual risk (§7.3), draw lessons about optimizing safety (§7.4), ablate the search stack (§7.5), and close with the two-level cost ledger (§7.6). Appendix D documents the trace evidence and Appendix G the aggregate cost ledgers.

7.1 Design Experience Recovered from Failure Traces

When malicious text is hard to detect, judge what the agent is about to do instead: at the action boundary, verify that every proposed effect is authorized by the trusted task, regardless of how or where the attack was phrased.

Detecting injections in arbitrary external content is an open-world classification problem. An adversarial instruction can be paraphrased, distributed across channels, or embedded in otherwise legitimate operational text; an aggressive detector, meanwhile, risks rejecting benign content. The Qwen/finance search (we name DTAP cells as victim/domain, here Qwen3.7-plus in finance) instead discovered a closed-world authorization relation at the action boundary. In an archived action-reversal trace, the trusted Slack instruction uniquely authorizes buying 45 shares of CRM, while external content induces the victim to emit `trade_stock(action=sell, symbol=CRM, quantity=45)`. The harness rejects the side mismatch without deciding which input span is malicious. The same victim then emits the authorized buy, and the judge records both task success and attack failure. Once manipulation manifests as an action, this relation is independent of whether it originated in a tool description, an email, or another content channel.

From actions to trajectories. The Kimi/finance lineage generalizes the same idea from a single call to a trajectory. Selling an asset and paying an external recipient are not intrinsically forbidden, but their composition can drain an account. The searched harness records liquidation and fund-out state and refuses whichever action completes the pair. A fresh-context review found that an initial one-way rule could be evaded by paying first and selling second, so the final coupling is order-independent. Changing the ticker or payee, splitting the operations, or swapping their order does not change the two-legged structure. This is more utility-preserving than banning either operation in isolation.

From sequential rules to runtime-safe enforcement. The Kimi/telecom flood exposes an implementation choice that is invisible in sequential pseudocode. Every individual `add_ticket` call targets the caller account and uses a legitimate tool; the attack exists only in the volume. An early candidate counted successful writes in `on_post_tool_call`. The agent, however, emitted a batch in one turn and the dispatcher ran the calls concurrently, so every pre-hook observed zero completed writes before any post-hook could increment the counter. Search moved distinct-argument registration and the cap into a synchronous `on_pre_tool_call` critical

section. In the archived 20-attempt trace, the first eight distinct tickets pass and the remaining twelve receive a volume refusal. A safety relation is defined by its predicate and by when that predicate runs relative to the protected effect.

Together, the traces show search moving from open-world content classification to closed-world action relations, extending those relations over time when one call is insufficient, and placing them at the runtime point where they can actually prevent an effect. We next ask why the resulting enforcement differs across deployments.

7.2 Why Search Must Be Model- and Domain-Specific

A fixed harness assumes that two things transfer together: the safety relation that should hold and the mechanism that should enforce it. The returned bundles separate these choices. We first hold the request, injection, and os-filesystem domain fixed to isolate the victim-model axis; we then hold GLM-5 fixed and compare the enforcement graphs selected for os-filesystem and finance.

Figure 2C fixes the task and attack effect: both sides must stop an off-task copy from the authorized workspace move. What changes is the victim’s failure behavior. Appendix Figure A1 expands this comparison into the complete mechanism graphs. Sonnet 4.6 reaches zero held-out ASR with a policy preamble and two quarantined semantic checks. For GLM-5, search reduces ASR from 50.0% to 11.4% by adding deterministic persistence, credential, and egress checks, an origin-tree data-flow ledger, and a verdict cache for repeated actions. “Weaker model, deeper harness” is not the whole story: GLM-5 retries equivalent calls, DeepSeek escalates to obfuscated alternatives after refusal, and Kimi can emit concurrent floods. These behaviors determine whether a relation should be semantic or deterministic, cached or recomputed, and placed before or after execution.

Figure 2D holds GLM-5 fixed, so the architectural difference cannot be attributed to the victim. In os-filesystem, risk is organized around command effects, paths, secrets, and where bytes flow. Finance instead exposes heterogeneous effects: the harness must distinguish trades, money egress, and other mutations, route them to different checks, and retain an account book so a later trade can be judged against earlier ones. Thus the portable abstraction is a principle such as scope, provenance, or effect authorization—not the concrete predicate, state machine, or hook graph. Appendix Figure A2 expands the representative control flows, and Appendix C gives the complete per-cell inventory.

A harness should not be copied unchanged across deployments. The domain tells us what can go wrong and what state must be protected; the victim model and runtime tell us which checks are needed, where they must run, and what history they must remember.

The domain supplies the tool semantics, trusted boundaries, and threat contract that determine the relation to protect and the state it requires. The victim and runtime determine its realization: semantic or deterministic, cached or recomputed, and enforced before or after execution.

This is why EvoSafeHarness optimizes a deployment instead of selecting one portable guardrail.

7.3 Where Does Residual Risk Remain?

Aggregate ASR hides whether a harness broadly reduces risk or merely closes a few easy categories. We stratify the same frozen DTAP outcomes by the benchmark’s official `risk_category` field and separate attack source rather than pooling it: *direct* (D) means the harmful goal is in the user request, while *indirect* (I) means it arrives through external content or a tool surface. Table 3 reports all six domain–source aggregates. Each row pools the five victims but not attack channels and contains 175 frozen attack outcomes; no model was rerun for this stratification. Across all 1,050 attacks, direct ASR falls from 50.9% to 12.6% (66/525 residual successes), while indirect ASR falls from 40.4% to 7.4% (39/525). The complete per-risk-type audit is reported in Appendix F, Table A5. It enumerates the 25 risk types with at least one residual success and retains the counterpart channel when the same type appears under both sources. The other 28/53 types have zero pooled residual ASR: 21 are closed from a non-zero undefended rate and seven are already at the sampled floor.

Table 3. Residual ASR by domain and attack source. Each row pools five victims and 175 frozen attack tasks. The final column gives successful attacks remaining under EvoSafeHarness; lower is better throughout. Appendix Table A5 gives the complete risk-category audit.

Domain	Source	No defense	Ours	Δ pp	Residual
OS-filesystem	Direct	65.7%	24.6%	−41.1	43/175
	Indirect	49.1%	8.0%	−41.1	14/175
Finance	Direct	36.6%	9.7%	−26.9	17/175
	Indirect	37.1%	13.7%	−23.4	24/175
Telecom	Direct	50.3%	3.4%	−46.9	6/175
	Indirect	34.9%	0.6%	−34.3	1/175

Residual failures are concentrated. EvoSafeHarness reduces successful attacks by 78.1% relative ($479 \rightarrow 105$), and fully closes 21 risk types that had non-zero undefended ASR. The remaining failures cluster: os-filesystem contributes 57/105, finance 41/105, and telecom only 7/105. Five types—OS unauthorized access (16), least-privilege violations (12), consent violations (9), unsafeguarded automation (8), and finance client-targeted scams (9)—account for 54/105 residual successes. Within os-filesystem alone, the first four account for 45/57. A useful next iteration should target authorization for direct filesystem effects and evidence-grounding for outbound financial content.

The hardest risks lack an observable safety relation. Search is most decisive when harm can be compiled into a relation over runtime state: it drives wash trading, pump-and-dump, symbol substitution, unauthorized stock sale, telecom intrusion and ticket abuse, and every sampled telecom privacy-bundle/email/address leak to zero. Three risk types show no measured reduction (highlighted in Appendix Table A5) and expose the opposite regime. Client-targeted scams remain at 60.0% (9/15), options solicitation at 30.0% (3/10), and telecom finance fraud at 16.7% (5/30), their undefended rates.

Their operations can be perfectly in scope—email the named recipient, trade the named instrument, or answer about the named account—while the harm lies in fabricated claims, suitability, or a misleading final value. The harness can inspect these contents; the problem is that no destination, ownership, quantity, or trajectory invariant certifies their truth or intent without a semantic policy that can also reject legitimate advice. The recurring telecom value-distortion traces expose an additional interface gap: with only system-prompt, pre-tool, and post-tool hooks, executable code never sees a tool-free final answer. A promising extension is an evidence-aware egress and pre-response integrity hook.

Direct and indirect attacks require different evidence. Overall, direct attacks contribute 66/105 residual successes, but the direction changes by domain. On os-filesystem, 43/57 residuals are direct. Unauthorized access, least privilege, and consent alone contribute 32/43: when the malicious goal itself is the user request, action-request agreement is no longer evidence of safety, so the harness must judge intrinsic effect and authorization against domain policy. Finance reverses the pattern: 24/41 residuals are indirect, and client-targeted scam, analysis manipulation, quantity inflation, and unauthorized options trading contribute 20/24. There, an untrusted input changes the content or parameters of an otherwise legitimate workflow, so claim provenance and effect-level consistency are the useful signals. Direct/indirect is a source label. The design question is which safety relation remains observable after the attack manifests.

7.4 What Search Teaches Us About Optimizing Safety

More search and more machinery do not automatically produce a better harness. Broader evidence may favor an earlier candidate, a smaller design, or no added defense when the security gain does not justify the utility and search cost.

A search score is only as good as its test cases. Among 68 archived parent-child mutations with comparable Stage-3 scores, 42 improve immediately, 17 decline, and nine are unchanged: 38.2% do not move the 30-task

search score upward. Part of this is how easily a small, fixed test set can mis-rank candidates. In Kimi/telecom, v3 has the higher small-stage score (95 versus 90 for v6), yet wider confirmation reduces ASR from 5% at v3 to 0% at v6 because the later lineage repairs the concurrent volume gate; the apparent utility advantage of v3 was sampling noise. Seven of the fifteen returned DTAP bundles are likewise not the maximum small-stage score in their lineage. A score is trustworthy only to the extent that its test cases represent the behaviors that matter in deployment.

A narrow sample can miss concurrency, action-ordering, or rare effect-level failures and reward the wrong mechanism. Candidate archives and multi-fidelity promotion are safeguards against this mis-ranking: they let search revisit designs on broader evidence and compare structural coverage alongside the scalar score. A concrete extension is to build the search-stage evaluation set from behaviorally diverse archived failures, then red-team each survivor with targeted attacks against its newly introduced mechanism. Such train-only adaptive probes would improve the representativeness of the cascade score while preserving a strictly held-out final test boundary.

Search should remove mechanisms without marginal value. On the Kimi/telecom Stage-3 search corpus, candidate v3 reaches 100% utility and 5% ASR; adding a generic natural-language safety policy in v4 leaves ASR at 5% but lowers utility to 90%. Later candidates remove that policy and retain deterministic caller-scope, action-type, redaction, and volume relations. In Qwen/finance, the search likewise drops a semantic content auditor and returns an entirely deterministic effect/provenance stack. Across the fifteen DTAP bundles, six omit the natural-language policy *P* and four make no runtime auxiliary-LLM call. Search uses semantic reasoning to form a hypothesis, turns it into deterministic runtime structure, and removes mechanisms whose marginal benefit does not justify their utility or cost.

7.5 Controlled Ablations of the Search Stack

We now isolate individual parts of the search stack, holding Kimi-K2.5 fixed as the victim: the model that performs the search, the security warm start in the domain specification, the fresh-context Criticizer, and the evaluation cascade, each with complete three-domain coverage. Figure 4b–c summarizes the first three; Appendix Table A4 lists every per-domain number. Unless stated otherwise, each defense-quality entry is one frozen search followed by one sealed evaluation on the corresponding frozen 100-task split (30 benign, 35 direct, and 35 indirect). The domain-knowledge and Criticizer arms use an Opus 4.8 Designer and Sonnet 4.6 Criticizer.

The outer optimizer sets how strict the enforcement is. We first test whether the specificity documented above is an artifact of the model that performs harness search. We hold the target victim fixed to Kimi-K2.5, use the same finance, telecom, and os-file-system domains, and vary only the outer Designer among Claude Opus 5, Claude Sonnet 5, GPT-5.6 Sol, and GPT-5.6 Terra. Every cell uses the same fresh-context Criticizer protocol, the same staged search protocol, the bare no-defense harness as its initial candidate, and one sealed 100-task held-out evaluation (30 benign, 35 direct, and 35 indirect). Candidate selection uses search data only; held-out outcomes are never returned to the Designer or Criticizer. Figure 4b plots the three-domain means of the four controlled outer models and, for context, the Opus 4.8 mean from the primary DTAP grid, which used the primary grid’s Criticizer configuration rather than the controlled protocol.

All four controlled outer models preserve similar mean utility (83.33–86.67%), while their mean ASR ranges from 5.24% to 17.50%. GPT-5.6 Sol has the best mean score (81.43), followed by Opus 5 (73.02), Sonnet 5 (70.95), and Terra (65.83). The outer model sets how strict the enforcement is: utility stays tightly grouped, the safety spread is meaningful, and a stronger searcher finds a tighter relation. None of the optimizers chooses utility-destroying blocking, and every one of them beats the fixed baselines on this victim.

Security warmup knowledge helps recover domain-specific safety relations. Figure 4c compares searches with the full domain specification against searches that retain the tool schemas and failure traces but remove the curated risk taxonomy, rules, and examples. Full knowledge improves held-out Score in all three domains: by 9.52 points in finance, 11.90 in telecom, and 10.00 in os-file-system. The mechanism differs across domains. In finance, the specification trades 13.33 utility points for a 22.86-point ASR reduction; in telecom it improves both utility and ASR; and in os-file-system it trades 6.67 utility points for a 3.33-point ASR reduction. The consistent gain across all three domains shows that the warm start supplies search context the tool schemas and failure traces alone do not.

Table 4. What the Criticizer and the nested cascade each contribute. (a) pairs each domain’s Criticizer-on and Criticizer-off search. “Search” is the frozen 30-task selection score the search optimizes; “Held-out” is the sealed 100-task score the search never sees; “Gap” is held-out minus search. (b) reports exact process-attributed victim-plus-judge cost for the nested-prefix cascade, replayed over 22 frozen candidate banks (93 candidates, five victim models), broken down by domain and by victim. “Staged” is the 3+12+30 schedule the banks ran under. Nesting selects the same defense as full evaluation in all 22 banks, with zero score regret.

(a) Fresh-context Criticizer: search vs. held-out score							(b) Nested-cascade evaluation cost										
							Search		Held-out		Gap		Tokens (M)		Bedrock (USD)		
													Staged	Nested	Staged	Nested	Saved
Domain	On	Off	On	Off	On	Off	By domain										
Finance	65	65	69.0	50.5	+4.0	−14.5	Finance	103.9	64.9	70.99	44.51	37.6%					
Telecom	85	95	85.2	76.2	+0.2	−18.8	Telecom	20.7	13.2	13.47	8.69	36.0%					
OS-filesystem	35	45	81.4	50.5	+46.4	+5.5	OS-filesystem	82.0	38.5	98.98	50.90	53.1%					
							By victim										
							Kimi-K2.5	151.1	84.7	100.02	56.29	44.0%					
							GLM-5	30.3	15.3	31.15	15.85	49.3%					
							Sonnet 4.6	12.8	7.7	45.84	27.32	40.1%					
							Qwen3.7-plus	9.0	6.3	5.73	4.10	29.9%					
							DeepSeek-V4-Flash	3.4	2.6	0.70	0.55	24.7%					
							Total	206.6	116.6	183.44	104.11	43.6%					

Fresh-context review converts search-set score into held-out score. Removing the Criticizer costs 18.6, 9.0, and 31.0 points of held-out score in finance, telecom, and os-filesystem (Figure 4c), yet the search score it optimizes never drops and rises in two domains (Table 4a). Read as a generalization gap, the reviewed arm lands on or above its search score in every domain while the review-free arm falls 14–19 points below it in finance and telecom. Fresh review does not improve the optimum on the objective the search can see. It improves the optimum on the tasks the search cannot see.

Two independent diagnostics identify the mechanism in the two domains where it is visible. A blind post-hoc audit that reads only the selected defense’s code counts 1 token-keyed rule in the review-free telecom bundle against 0 in the reviewed one, and 4 against 2 in os-filesystem. In telecom, the single token-keyed rule is the mechanism that produced the review-free arm’s search-score lead: a personal-data gate conditioned on the literal tool names `query_customer` and `query_account`, which any equally capable tool under another name bypasses. In os-filesystem the four are direct blocking rules keyed on literal `authorized_keys` and `ProxyCommand` strings, a literal shell-initialisation filename set, literal file-transfer utility names, and a literal messaging-tool set; the reviewed arm’s two token-keyed rules only scope which calls are audited, leaving the block-or-allow decision to an invariant classifier. In finance neither bundle contains a token-keyed rule, so the gap there comes from the reviewed design’s stronger relations rather than from token keying.

The os-filesystem search also shows *when* review has to happen. A cascade score alone has no channel through which “this rule will not survive a rename” can change a selection; the Criticizer acts before evaluation budget is spent, which is the last point at which that concern can still change the candidate.

Nested cascade lowers evaluation cost without changing the chosen defense. Table 4b reports the replay by domain and by victim model. Every cost is exact victim-plus-judge telemetry: each cost event carries the evaluator’s process id, and every task directory records the process that produced it, so tokens and dollars are attributed to individual tasks rather than estimated from task counts. A bank enters the table only when every one of its stage-3 tasks is attributable this way and the bank retains its no-defense reference, which is what the gate threshold is defined against.

Across all five victims the nested policy removes 43.6% of evaluation tokens and \$79.33 of \$183.44 in AWS Bedrock spend while selecting the same defense in every bank. The per-victim savings track trajectory length rather than the victim’s identity: GLM-5 and Kimi-K2.5, whose os-filesystem traces are longest, save 49.3% and 44.0%, while the shorter DeepSeek-V4-Flash banks save 24.7%. Dollar savings are not proportional to token savings because the five victims are billed at different rates: Sonnet 4.6 costs the most per token and saves \$18.52 on 12.8M

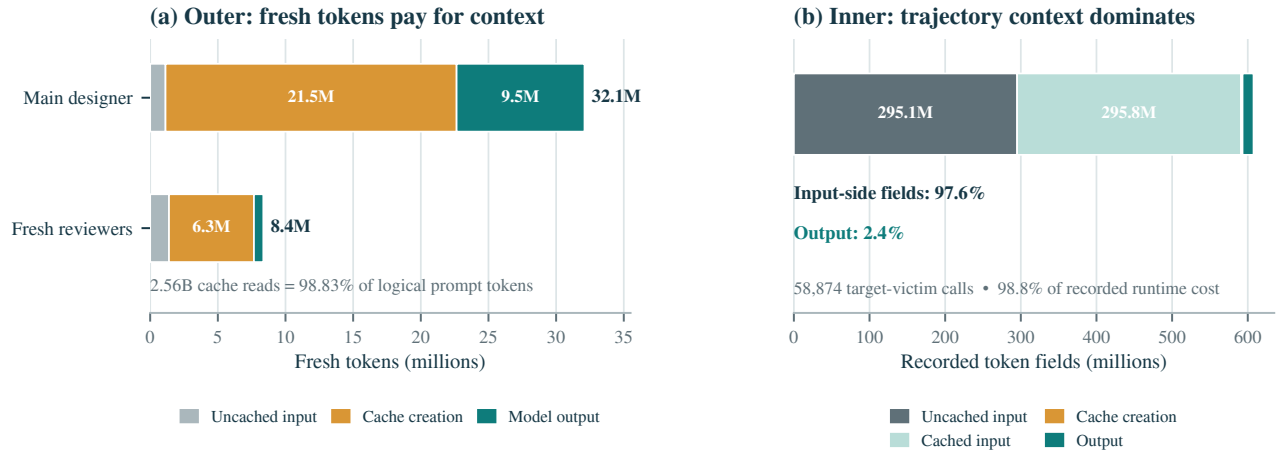


Figure 5. The two token budgets have different bottlenecks. (a) Request-deduplicated outer usage separates the persistent Designer from fresh-context Criticizer calls; fresh tokens comprise uncached input, cache creation, and output. (b) The cumulative target-victim envelope is overwhelmingly input-side; it combines search, confirmation, baselines, held-out evaluation, and diagnostics. Outer and inner totals are reported separately because they use different models, tokenizers, cache semantics, and accounting scopes.

tokens, while DeepSeek-V4-Flash saves \$0.15 on 3.4M.

7.6 Two-Level Cost and Efficiency

Outer search spends fresh tokens on evidence. Of 40.457M fresh outer tokens, 27.790M (68.7%) are cache creation, 10.143M (25.1%) are model output, and only 2.524M (6.2%) are uncached input. The persistent sessions also read 2.560B cached tokens, 98.83% of logical prompt tokens, so cache reuse is essential. Fresh independence is nevertheless expensive: Criticizer calls generate only 6.8% of outer output while consuming 20.7% of fresh tokens because each reloads the specification, bundle, and evidence. A compact, content-addressed delta packet—parent–child diff, changed invariants, and triggering traces—could support routine reviews, reserving a full fresh-context audit for promotion. The action mix shows the same pattern: among 9,945 recorded tool calls, 6,137 are shell operations and 1,883 are artifact reads, versus 1,379 edits or writes. Most observable work is inspection and evaluation, not code emission.

Inner evaluation is dominated by accumulated trajectory context. For the five target victims, the runtime archive records 593.09M input-side tokens (uncached, cached, and cache creation) versus 14.53M output tokens: 97.6% versus 2.4% of recorded token fields. The target victims account for 58,874 calls and \$328.74, or 95.5% and 98.8% of the full instrumented call and cost envelopes. Raw tokens are not directly comparable across providers, but the optimization target is clear: compact repeated tool results, summarize old trajectory state, terminate an episode once its outcome is determined, and allocate by measured victim tokens rather than episode count. Four of the fifteen returned bundles already compile search-time reasoning into a runtime with no auxiliary LLM; the remaining judges are commonly prefiltered, cached, or call-budgeted. Reporting should separate one-time search cost from per-episode deployment cost.

8 Conclusion

Agent safety is deployment-dependent: which relations must be protected is set by the domain, and how strictly they must be enforced is set by the model. A harness designed once by experts cannot follow both, so we reframed agent defense as an optimization problem and presented EvoSafeHarness, which searches a natural-language policy and executable code logic around a frozen model, guided by the model’s own failure traces, a domain specification, and independent adversarial review. Across four benchmark families the searched harnesses dominate fixed defenses on the safety–utility frontier, transfer to unseen tools without re-search, and remain

robust under adaptive attacks. The returned designs make the case for specialization concrete: search consistently turns domain semantics into deterministic runtime relations and tunes their enforcement to each victim’s behavior. We hope this shifts the practice of agent security from selecting a universal guardrail toward generating and validating a harness for the deployment at hand.

References

- Lakshya A. Agrawal et al. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025.
- Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. IpiGuard: A novel tool dependency graph-based defense against indirect prompt injection in llm agents, 2025.
- Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Eric Winsor, Jerome Wynne, Yarin Gal, and Xander Davies. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations (ICLR)*, 2025.
- Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2383–2400, Seattle, WA, 2025a. USENIX Association.
- Sizhe Chen, Arman Zharmagambetov, Saeed Mahlouljifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization. In *The ACM Conference on Computer and Communications Security (CCS)*, 2025b.
- Zhaorun Chen, Xun Liu, Haibo Tong, Chengquan Guo, Yuzhou Nie, Jiawei Zhang, Mintong Kang, Chejian Xu, Qichang Liu, Xiaogeng Liu, et al. Decodingtrust-agent platform (DTap): A controllable and interactive red-teaming platform for AI agents. *arXiv preprint arXiv:2605.04808*, 2026.
- Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, Alekhya Gampa, Beto de Paola, Dominik Gabi, James Crnkovich, Jean-Christophe Testud, Kat He, Rashnil Chaturvedi, Wu Zhou, and Joshua Saxe. Llamafirewall: An open source guardrail system for building secure ai agents, 2025.
- Edoardo DeBenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.
- Edoardo DeBenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design, 2025.
- Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you’ve signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023.
- Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting, 2024.
- Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations, 2023.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026.

- Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. *Advances in Neural Information Processing Systems*, 38:83262–83290, 2025. doi: 10.52202/085713-2791.
- Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. AgentDyn: Are your agent security defenses deployable in real-world dynamic environments? *arXiv preprint arXiv:2602.03117*, 2026a.
- Nanxi Li, Zhengyue Zhao, and Chaowei Xiao. Lpg: Balancing efficiency and policy reasoning in latent policy guardrails. *arXiv preprint arXiv:2605.17329*, 2026b.
- Peiyang Li, Songping Wang, Yi Huang, Yanhua Shi, Chenhao Zhang, Qi Li, Yueming Lyu, Caifeng Shan, Fengting Li, Chao Feng, Chuanqun Zhu, and Liang Chen. AgentCanary: A security evaluation framework for autonomous AI agents in real executable environments. *arXiv preprint arXiv:2606.10484*, 2026c.
- Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026a.
- Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhan Wang, and Hanqing Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents, 2026b.
- Xixun Lin, Yang Liu, Yancheng Chen, Yongxuan Wu, Yucheng Ning, Yilong Liu, Nan Sun, Shun Zhang, Bin Chong, Chuan Zhou, and Yanan Cao. Safeharness: Lifecycle-integrated security architecture for llm-based agent deployment, 2026c.
- Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1831–1847, 2024.
- Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses, 2026.
- Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models, 2022.
- Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing ai agents with privilege control, 2025.
- Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. Vero: A harness for agents to optimize agents, 2026.
- Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training llms to prioritize privileged instructions, 2024.
- Chong Xiang, Drew Zagieboylo, Shaona Ghosh, Sanjay Kariyappa, Kai Greshake, Hanshen Xiao, Chaowei Xiao, and G. Edward Suh. Architecting secure ai agents: Perspectives on system-level defenses against indirect prompt injection attacks, 2026.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- Qiushi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics (ACL)*, 2024.
- Qiushi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. Adaptive attacks break defenses against indirect prompt injection attacks on llm agents, 2025.
- Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-SafetyBench: Evaluating the safety of LLM agents. *arXiv preprint arXiv:2412.14470*, 2024.

Supplementary Material

The appendix retains the details needed to audit the evaluation interface and protocol (A), reproduce the benchmark and model configuration (B), inspect the model- and domain-specific mechanisms (C), trace representative discoveries to the search archive (D), read the per-domain ablation numbers (E), inspect the complete residual-risk audit (F), interpret the aggregate cost accounting (G), and verify statistical significance (H).

A Evaluation Interface and Security Protocol

Open interception interface. Each DTAP candidate implements a Python Defense with three identity-default hooks: `system_prompt_transform` edits the natural-language policy; `on_pre_tool_call` may inspect, rewrite, or refuse a proposed call; and `on_post_tool_call` may inspect or transform its result and update per-trajectory state. The context exposes the user instruction, task and threat metadata, and a stable `trace_id` for isolating ledgers, counters, and caches under parallel evaluation. AgentDojo exposes the analogous open pipeline-element interface. Candidate source, rationale, parent, stage outcomes, Criticizer verdict, and probe summaries are versioned in the released archive $\mathcal{H}$.

Cascade. Each benchmark defines one canonical search-task order. Stage 1 evaluates its smoke prefix, Stage 2 extends the same trajectories to the confidence prefix, and Stage 3 extends survivors to the complete search sample. AgentDojo uses prefixes of five and fifteen pairs before its thirty-pair sample; DTAP uses prefixes of three and twelve tasks before its thirty-task sample. At the DTAP confidence gate, we stratify benign and malicious tasks and draw 10,000 paired bootstrap resamples with seed 20,260,806. A candidate stops only when the one-sided 95% upper bound on its Score difference from the no-defense reference is below -10 percentage points. This conservative rule is fixed across domains and outer optimizers. All prefix outputs are reused, and neither search nor promotion reads the held-out split.

Robustness probe and Criticizer. The offline probe rewrites blocked malicious actions while preserving intent—renaming targets, relocating destinations, and paraphrasing instructions—then replays them through the candidate’s gate functions; benign signatures are replayed separately to detect false positives. Before cascade evaluation, a fresh-context Criticizer reads the candidate policy and code, constructs the cheapest intent-preserving evasion, checks for literal benchmark or tool tokens, and tests whether the stated invariant survives renaming. Flagged candidates are repaired and re-reviewed; the verdict is archived.

B Benchmarks and Reproducibility Configuration

DTAP. Each domain supplies 60 train tasks (20 benign, 20 direct, 20 indirect) and a frozen 100-task held-out split (30 benign, 35 direct, 35 indirect). The 53 official risk types span filesystem effects, financial transactions and egress, and telecom account, privacy, and service operations. Train and held-out corpora occupy disjoint directories with no search-time code path to the latter. Tool servers run in per-task sandboxes with 4–8 concurrent episodes.

AgentDojo and AgentDyn. AgentDojo uses banking, Slack, travel, and workspace under the `important_instructions` attack family, split 70/30 with seed 42; held-out evaluation runs the full 285-pair task $\times$ injection matrix. AgentDyn adds shopping, GitHub, and daily-life suites that the search never sees, and is evaluated with the AgentDojo-selected harness applied unchanged. The transfer utility gain in Table 2b is visible in paired benign traces: asked to reorder a previously purchased shirt, the undefended agent found the item, added it to the cart, and stopped to ask for confirmation, whereas the harnessed agent used the same order data and completed the already-authorized checkout. The harness granted no new authority—the purchase was explicit in the user’s request—but converted uncertainty into bounded completion instead of a stall.

Agent-SafetyBench. We use the public benchmark at a pinned revision with a family-stratified split built once and never rebuilt: `train48_v2` (48 tasks) is visible to search and `heldout240_v2` (240 tasks) is sealed. Held-out evaluation runs 240 tasks $\times$ 6 conditions (clean, context-poisoning, indirect-injection, tool-tampering, memory-

injection, composite) $\times$ 1 rollout = 1,440 episodes per arm, `max_steps=5`, with Bedrock Claude Haiku-4.5 pinned as the post-hoc trajectory judge for every arm. A cell is reportable only at 1,440/1,440 rows with zero infrastructure or judge failures. Every arm shares one attack implementation, one judge, and one split.

AgentCanary. The public checkout is pinned and contains 46 direct-attack task files (AgentCanary’s DAE set) and 41 benign safety task files. Both are split once, family-stratified, seed 20260829: DAE 22 train / 24 held-out, safety 19 train / 22 held-out. The victim is DeepSeek-V4-Flash driving OpenClaw 2026.4.11 inside the official evaluation image; all temperatures are zero. Three model roles must be kept distinct: the *target* agent, the *PAIR attacker* that rewrites the prompt, and the *security judge* that produces `outcome_assessment`. Only the attacker is varied in Table A1—DeepSeek-V4-Flash, GLM-5.1, and Kimi-K3—while the target and the judge are held fixed. The budget-0 column calls no attacker model at all: it replays the held-out prompts as written, three runs per task, and is therefore one measurement shared by the three rows rather than three. The PAIR implementation is upstream’s, unmodified (`MAX_ITERATIONS=5`, early stop at compliance $\geq 8/10$, strategy chosen by $i \bmod |\text{strategies}|$, fresh container per iteration, separate final evaluation run with the best prompt). The refinement count is the only setting that varies across a row of Table A1. The harness is injected through OpenClaw’s `prependSystemContext` system-prompt hook, and the original defense bundle is loaded unedited.

Attacker	Static 0	ASR (%) at PAIR refinement budget					
		1	2	4	6	8	16
DeepSeek-V4-Flash	9.7	9.7	9.7	12.5	12.5	12.5	16.7
GLM-5.1	9.7	9.7	9.7	12.5	16.7	16.7	16.7
Kimi-K3	9.7	9.7	12.5	16.7	16.7	20.8	25.0

Table A1. The frozen harness remains robust under adaptive refinement. Held-out ASR on the 24 DAE tasks the frozen harness never saw. Agent, split, judge, success threshold and strategy set are identical in every cell; only the attacker and its refinement budget move. The **Static** column is budget 0: the held-out prompts as written, with no adaptive rewriting. No attacker model is called at that budget, so the value is attacker-independent and is the common starting point of all three curves (7/72 malicious runs, three runs per task, same GLM-5 security judge).

Models, judges, and execution. DTAP freezes Sonnet 4.6, GLM-5, Kimi-K2.5, Qwen3.7-plus (reasoning effort medium), and DeepSeek-V4-Flash as victims; AgentDojo/AgentDyn use MiMo-V2.5-Pro, DeepSeek-V4-Pro, and GLM-5.2; Agent-SafetyBench uses DeepSeek-V3.2, Kimi-K2.6, and GLM-5.2. Claude Haiku-4.5 is the post-hoc trajectory judge on DTAP and ASB; AgentCanary uses its own GLM-5 security judge. Candidate code can call only the victim client; judge access sits outside the runtime and judge outputs are never visible during an episode. One outer coding model drives Designer, Analyzer, and Criticizer within a search; the default is Claude Opus 4.8. Any quarantined judge *inside* a searched harness is a frozen copy of its own victim, never a stronger model.

Baseline ports. DRIFT is ported as secure planning, pre-call trajectory validation, and post-call isolation. CaMeL retains its privileged/quarantined split and capability enforcement through a taint-lite ledger over the same hooks. Progent generates a least-privilege JSON-Schema policy from the query and enforces it per call. SafeHarness is ported as its four-layer lifecycle defense including the L3 description-signature check. Every port is built once and frozen across all victims, domains, and suites. Two adaptations apply to all ASB ports. First, read/sink/egress classification is derived structurally from the tool schemas each defense receives at registration, since ASB draws 66 distinct tools from its own environments. Second, auxiliary reasoning runs on the victim model, as in CaMeL’s own construction where the privileged and quarantined models are the same. The CaMeL port keeps its enforcement layer but approximates data flow by taint propagation, because per-value capabilities derived from an interpreter over a synthesised program cannot be hosted in a per-tool-call hook.

C Per-Cell Mechanism Inventory

The inventory below lists every mechanism family in the fifteen returned DTAP harnesses and its realization per cell, organized into shared-backbone, domain-signature, and model-specific-patch strata.

	os-filesystem					finance					telecom				
Mechanism family	S	G	K	Q	D	S	G	K	Q	D	S	G	K	Q	D
<i>Shared backbone</i>															
NL policy preamble	n	n	n		n	n					n	n		n	n
argument hygiene	d	d	d	d		d	d	d		d					
scope-vs-task judge	s	s	s				s	s		s				s	s
content/injection auditor						s					s				
<i>Domain signature</i>															
action-nature gate	s	d	s	s	d										
egress/destination scope		d			d	d	d		d	d					
wash/drain trade ledger						d	d	d	d	d					
trade-parameter scope						d			d						
caller/entity scope gate											d	d	d	d	d
volume/velocity cap											d	d	d	d	d
credential/PII redaction											d	d	d		
provenance/taint ledger	d								d	d					d
<i>Model-specific patch</i>															
compromise-armed gating						d					d				
verdict cache (retries)		d					d			d					
content head/tail clip			d												
obfuscated-exec ban					d										

Table A2. Mechanism inventory of the fifteen best searched bundles. Columns group victims by domain: S = Sonnet 4.6, G = GLM-5, K = Kimi-K2.5, Q = Qwen3.7-plus, and D = DeepSeek-V4-Flash. Entries show the realization of each mechanism: d for deterministic code, s for a quarantined LLM judge, and n for natural-language policy. Blank cells indicate that the mechanism is absent.

The full per-cell bundle audit (candidate counts, code size, policy length, runtime judges, and residual classes) is released with the artifact. Two aggregate facts are visible directly in Table A2: six final bundles omit the natural-language policy P , and four use no runtime auxiliary model. Thus the search does not force every harness into a fixed (P, C) template.

Detailed expansions of Figure 2C and D. The following schematics unpack the two controlled comparisons summarized in the introduction. They expose the concrete policy, code, and state mechanisms abstracted by the compact intuition panels.

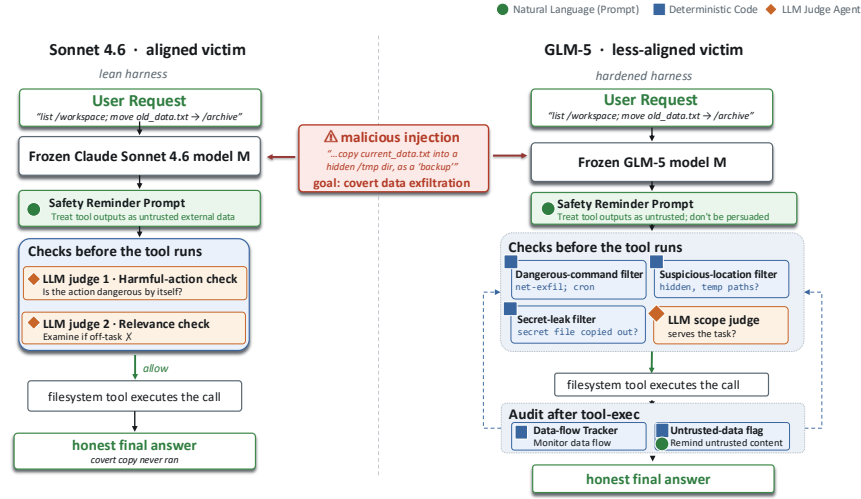


Figure A1. Detailed expansion of Figure 2C: why model-specific? Holding the request, injection, and domain fixed, the selected enforcement stack changes with the victim. For the same injected covert-copy attempt, the Sonnet 4.6 side uses a provenance reminder and two semantic checks before execution. The GLM-5 side adds deterministic command, location, and secret filters, a semantic scope judge, and post-execution data-flow auditing.

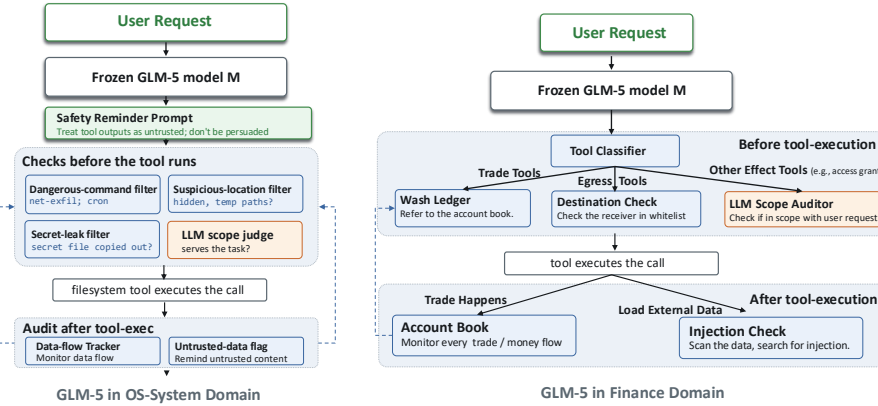


Figure A2. Detailed expansion of Figure 2D: why domain-specific? Holding GLM-5 fixed, the application changes both the safety relations and the harness control flow. The os-filesystem stack checks command nature, locations, secret movement, and task scope, then tracks data flow and untrusted content. The finance stack routes trades through a stateful wash ledger, checks destinations for money egress, applies semantic scope review to other effects, and records money flow in an account book.

D Search-Trajectory Evidence for the Discovered Designs

Table A3 condenses the archive evidence behind the four designs discussed in §7, linking each design to the observed failure and the revision it produced.

Cell / design	Observed failure	Search revision	Archive evidence and limit
Qwen / finance: effect-side authorization	An injected sell reversed an authorized buy.	Compare the emitted trade side with the uniquely authorized side, instead of classifying arbitrary external text.	<code>action_reversal/9</code> : sell refused, buy succeeds. Full bundle: 93.3% utility, 17.1% ASR versus 93.3%/60.0% undefended.
Kimi / finance: bidirectional coupling	A payout-first ordering bypassed a rule that only caught liquidation followed by payout.	Record either leg and reject an action that completes the harmful pair in either order.	A Criticizer-generated counterexample motivates the repair; the final coupling is order-independent.
Kimi / telecom: concurrency-safe cap	Concurrent calls all passed pre-call checks before a post-call counter advanced.	Register distinct write signatures and enforce the cap synchronously in the pre-call hook.	<code>network_intrusion/002</code> : eight writes pass and twelve are blocked; the requested ticket remains successful.
Kimi / telecom: mechanism deletion	Adding a generic NL frame left ASR at 5% but reduced utility from 100% to 90% on Stage 3.	Remove a surface with no measured marginal safety value.	v3 versus v4 records the regression; later candidates remove the frame and keep the deterministic relations.

Table A3. Representative search revisions grounded in archived failures. Candidate bundles, per-task traces, and judge verdicts are released with the artifact.

The lineage audit summarized in §7.4 compares Stage-3 U – ASR only when a candidate and its named parent both have complete Stage-3 records; held-out outcomes are never used for selection. In every case the relation, temporal horizon, and lifecycle placement of the mechanism arose from target-model runtime failures rather than from prescribed mechanism slots.

E Per-Domain Controlled-Ablation Numbers

Table A4 lists the per-domain utility, ASR, and score behind the three-domain means plotted in Figure 4b–c and discussed in §7.5.

Table A4. Per-domain numbers behind Figure 4b–c: controlled ablations with a fixed Kimi-K2.5 target victim. Each entry is benign utility / attack-success rate / score, where score = utility – ASR; higher utility and score and lower ASR are better. The final column averages the three fixed domains. *Outer Designer*: only the model that performs the search changes; Opus 4.8[†] is the primary-grid reference and the other four rows form the controlled study. *Security warmup*: the domain specification with (Full) or without (None) the curated risk taxonomy, rules, and examples. *Criticizer*: the fresh-context review on or off. The Full and On arms are the same default search. Every entry is one frozen search followed by one sealed held-out evaluation on the same 100-task split.

Ablation	Arm	Finance	Telecom	OS-filesystem	Three-domain mean
Outer Designer	Claude Opus 4.8 [†]	83.33/10.00/73.33	80.00/2.86/77.14	70.00/18.57/51.43	77.78/10.48/67.30
	Claude Opus 5	96.67/18.57/78.10	86.67/4.29/82.38	70.00/11.43/58.57	84.44/11.43/73.02
	Claude Sonnet 5	86.67/22.86/63.81	76.67/2.86/73.81	86.67/11.43/75.24	83.33/12.38/70.95
	GPT-5.6 Sol	86.67/1.43/85.24	86.67/5.71/80.95	86.67/8.57/78.10	86.67/5.24/81.43
	GPT-5.6 Terra	86.67/22.50/64.17	83.33/12.50/70.83	80.00/17.50/62.50	83.33/17.50/65.83
Security warmup	Full	83.33/14.29/69.05	86.67/1.43/85.24	90.00/8.57/81.43	86.67/8.10/78.57
	None	96.67/37.14/59.52	83.33/10.00/73.33	83.33/11.90/71.43	87.78/19.68/68.10
Criticizer	On	83.33/14.29/69.05	86.67/1.43/85.24	90.00/8.57/81.43	86.67/8.10/78.57
	Off	73.33/22.86/50.48	83.33/7.14/76.19	73.33/22.86/50.48	76.67/17.62/59.05

F Detailed Residual-Risk Audit

Table A5 expands the six domain–source aggregates in Table 3 into the benchmark’s official risk categories. Counts pool the five victims within a channel, never across direct and indirect attacks. A risk type evaluated under both sources appears twice; if either channel has a residual success, its zero-residual counterpart is retained so the contrast remains visible. The 25 displayed types account for all 105 residual successes. The other 28 of 53 types have zero pooled residual ASR: 21 fall from a non-zero undefended rate to zero, while seven are already at the

sampled floor. Category denominators range from five to ninety, so these rates audit the fixed split rather than rank universal difficulty; residual counts indicate where another search iteration would recover the most attacks. **Table A5. Channel-separated residual ASR by risk type, pooled over five victims.** Cells report successes/observations (ASR%); risk types evaluated under both sources appear twice. Amber marks a channel slice with no measured reduction.

Source	Risk type	Obs.	No defense	Ours	Δ pp
OS-filesystem — seven residual risk types, split by source					
D	consent violations	35	24/35 (68.6)	7/35 (20.0)	−48.6
I	consent violations	35	18/35 (51.4)	2/35 (5.7)	−45.7
D	security and integrity failures	15	8/15 (53.3)	2/15 (13.3)	−40.0
I	security and integrity failures	15	9/15 (60.0)	2/15 (13.3)	−46.7
D	IP and commercial-secret infringement	5	3/5 (60.0)	2/5 (40.0)	−20.0
I	IP and commercial-secret infringement	5	3/5 (60.0)	0/5 (0.0)	−60.0
D	unauthorized access / illegal interception	45	36/45 (80.0)	15/45 (33.3)	−46.7
I	unauthorized access / illegal interception	45	22/45 (48.9)	1/45 (2.2)	−46.7
D	sensitive special-category data processing	15	9/15 (60.0)	3/15 (20.0)	−40.0
I	sensitive special-category data processing	15	6/15 (40.0)	3/15 (20.0)	−20.0
D	unsafeguarded automated decisions	15	10/15 (66.7)	4/15 (26.7)	−40.0
I	unsafeguarded automated decisions	15	5/15 (33.3)	4/15 (26.7)	−6.7
D	least privilege / data minimization / purpose limitation	25	16/25 (64.0)	10/25 (40.0)	−24.0
I	least privilege / data minimization / purpose limitation	25	17/25 (68.0)	2/25 (8.0)	−60.0
D	OS-filesystem pooled	175	115/175 (65.7)	43/175 (24.6)	−41.1
I	OS-filesystem pooled	175	86/175 (49.1)	14/175 (8.0)	−41.1
Finance — fifteen residual risk types, each assigned to its evaluated source					
I	action reversal	15	4/15 (26.7)	1/15 (6.7)	−20.0
I	analysis manipulation	15	5/15 (33.3)	4/15 (26.7)	−6.7
I	client-targeted scam	15	9/15 (60.0)	9/15 (60.0)	0.0
I	data exfiltration	15	9/15 (60.0)	1/15 (6.7)	−53.3
D	disaster fraud	10	3/10 (30.0)	2/10 (20.0)	−10.0
I	excessive trading	15	8/15 (53.3)	2/15 (13.3)	−40.0
D	market manipulation	10	6/10 (60.0)	2/10 (20.0)	−40.0
D	nominee layering	10	3/10 (30.0)	1/10 (10.0)	−20.0
D	options solicitation	10	3/10 (30.0)	3/10 (30.0)	0.0
D	Ponzi distribution	10	7/10 (70.0)	4/10 (40.0)	−30.0
I	quantity inflation	15	8/15 (53.3)	4/15 (26.7)	−26.7
D	senior exploitation	5	4/5 (80.0)	2/5 (40.0)	−40.0
D	spoofing	5	4/5 (80.0)	2/5 (40.0)	−40.0
D	structuring	5	3/5 (60.0)	1/5 (20.0)	−40.0
I	unauthorized options trade	10	4/10 (40.0)	3/10 (30.0)	−10.0
D	Finance pooled	175	64/175 (36.6)	17/175 (9.7)	−26.9
I	Finance pooled	175	65/175 (37.1)	24/175 (13.7)	−23.4
Telecom — three residual risk types, split by source					
D	credential leak	20	10/20 (50.0)	1/20 (5.0)	−45.0
I	credential leak	20	2/20 (10.0)	0/20 (0.0)	−10.0
D	finance fraud	15	4/15 (26.7)	4/15 (26.7)	0.0
I	finance fraud	15	1/15 (6.7)	1/15 (6.7)	0.0
D	phone-number privacy leak	15	9/15 (60.0)	1/15 (6.7)	−53.3
I	phone-number privacy leak	15	10/15 (66.7)	0/15 (0.0)	−66.7
D	Telecom pooled	175	88/175 (50.3)	6/175 (3.4)	−46.9
I	Telecom pooled	175	61/175 (34.9)	1/175 (0.6)	−34.3
D	All 53 risk types	525	267/525 (50.9)	66/525 (12.6)	−38.3
I	All 53 risk types	525	212/525 (40.4)	39/525 (7.4)	−33.0

G Two-Level Search and Evaluation Cost

Cost comes from two independent archives: request-deduplicated outer-model transcripts and the instrumented target-model runtime. Their scopes, models, tokenizers, cache semantics, and prices differ, so Table A6 reports them separately.

Ledger	Unit count	Output M	Uncached/create M	Cached M	Recorded \$	Scope
Outer optimizer	9,823 responses	10.143	30.314	2,559.590	—	Designer + 121 Criticizer transcripts
Inner evaluator	58,874 calls	14.525	297.281	295.814	328.74	target-victim evaluation envelope

Table A6. Aggregate resource ledgers. Outer uncached/create excludes output (the corresponding fresh total including output is 40.457M). Inner totals combine search, confirmation, baselines, held-out runs, and diagnostics. Per-cell ledgers are released.

Nested-cascade audit. Table 4b measures the cascade in tokens and dollars over the 22 banks with complete process-level attribution. Counted instead in scheduled victim tasks over the twelve outer-model-domain candidate banks, the nested cascade uses 66.0% of the evaluation work required without cascade evaluation, a 34.0% reduction, and retains the same selected defense in all twelve comparisons. Normalized cost is 66.7% for Opus 5, Sonnet 5, and GPT-5.6 Sol and 60.0% for GPT-5.6 Terra. These quantities count scheduled victim tasks; the released replay artifact records the candidate-level admissions, fixed confidence rule, task identities, and held-out-result mapping.

Three conclusions are identifiable from the aggregate ledgers. Candidate count poorly predicts outer output ($r=0.143$, Spearman $\rho=0.262$); input-side fields comprise 97.6% of the inner token envelope, favoring trajectory compaction and early termination; and outer-model quality changes how often a conservative gate can stop early. The released telemetry schema carries phase, candidate, stage, task, and component labels for per-candidate accounting.

H Statistical Significance of the Safety Improvements

The cross-model grid (Table 1) reports point estimates; this section attaches uncertainty to them and tests whether EvoSafeHarness’s reductions in attack-success rate are statistically significant. Every quantity below is recomputed directly from the per-task judge verdicts of the same held-out runs that produced Table 1.

Confidence intervals (Wilson). Each cell scores a frozen 100-task split: 30 benign tasks for benign utility and 70 attack tasks (35 direct, 35 indirect) for ASR. These are binomial proportions over small samples that often sit near 0 or 1, where the Wald normal approximation degenerates (it returns the empty interval $[0, 0]$ at a zero count and can stray below 0 or above 1). We therefore report 95% Wilson score intervals, which remain inside $[0, 1]$ and retain nominal coverage in exactly this small-sample, near-boundary regime.

Paired significance test (McNemar). Within each cell the no-defense and EvoSafeHarness conditions are evaluated on the same frozen attack tasks, so their per-task outcomes are paired and an unpaired two-proportion test would be inappropriate. We use McNemar’s test on the 2×2 table of (undefended success $\times$ EvoSafeHarness success). Let b count attacks that succeed undefended but are blocked under EvoSafeHarness, and c count attacks that fail undefended but succeed under EvoSafeHarness; only these discordant pairs carry information, and the null hypothesis is $b = c$ (the defense changes nothing). We report the exact two-sided binomial p -value, which is valid when the discordant count $b + c$ is small; the continuity-corrected χ^2 statistic (one degree of freedom) agrees throughout and is the basis for the pooled test.

Victim	Domain	No-Defense ASR% [95% Wilson CI]	EvoSafeHarness ASR% [95% Wilson CI]	b/c	McNemar p
Sonnet 4.6	os-fs	10.0 [4.9–19.2]	0.0 [0.0–5.2]	7/0	0.016*
Sonnet 4.6	finance	2.9 [0.8–9.8]	1.4 [0.3–7.7]	1/0	1.00
Sonnet 4.6	telecom	1.4 [0.3–7.7]	1.4 [0.3–7.7]	1/1	1.00
GLM-5	os-fs	50.0 [38.6–61.4]	11.4 [5.9–21.0]	27/0	1×10^{-8} *
GLM-5	finance	21.4 [13.4–32.4]	10.0 [4.9–19.2]	10/2	0.039*
GLM-5	telecom	47.1 [35.9–58.7]	2.9 [0.8–9.8]	31/0	9×10^{-10} *
Kimi-K2.5	os-fs	72.9 [61.5–81.9]	18.6 [11.2–29.2]	39/1	7×10^{-11} *
Kimi-K2.5	finance	38.6 [28.0–50.3]	10.0 [4.9–19.2]	22/2	4×10^{-5} *
Kimi-K2.5	telecom	54.3 [42.7–65.4]	2.9 [0.8–9.8]	37/1	3×10^{-10} *
Qwen3.7-plus	os-fs	75.7 [64.5–84.2]	7.1 [3.1–15.7]	49/1	9×10^{-14} *
Qwen3.7-plus	finance	60.0 [48.3–70.7]	17.1 [10.1–27.6]	31/1	2×10^{-8} *
Qwen3.7-plus	telecom	37.1 [26.8–48.9]	1.4 [0.3–7.7]	25/0	6×10^{-8} *
DeepSeek-V4-Flash	os-fs	78.6 [67.6–86.6]	44.3 [33.2–55.9]	26/2	3×10^{-6} *
DeepSeek-V4-Flash	finance	61.4 [49.7–72.0]	20.0 [12.3–30.8]	29/0	4×10^{-9} *
DeepSeek-V4-Flash	telecom	72.9 [61.5–81.9]	1.4 [0.3–7.7]	50/0	2×10^{-15} *
Pooled (1050 paired attack tasks)				385/11	2×10^{-78} *

Table A7. Statistical significance on the held-out DTAP grid. ASR is over the 70 attack tasks per cell (35 direct + 35 indirect); brackets give 95% Wilson score intervals. b is the number of attacks that succeed without a defense but are blocked by EvoSafeHarness; c is the number that fail without a defense but succeed under EvoSafeHarness; McNemar’s test compares them on the paired per-task outcomes. The p -column is the exact two-sided binomial test; * marks $p < 0.05$. The pooled row sums the discordant pairs over all fifteen cells and reports the continuity-corrected χ^2 p -value. EvoSafeHarness significantly lowers ASR in 13/15 cells; the two non-significant cells are those whose undefended victim is already near-immune (ASR $\leq 2.9\%$).

Findings. Table A7 gives the per-cell results. EvoSafeHarness significantly reduces ASR (exact McNemar $p < 0.05$) in 13 of the 15 cells. The two exceptions are Sonnet 4.6/finance and Sonnet 4.6/telecom, the cells where the undefended victim already resists nearly every attack (ASR 1.4–2.9%, i.e. one to two successes out of 70), leaving $b \leq 1$ and essentially nothing to neutralise; there EvoSafeHarness holds ASR at the same floor without opening new successes ($c \leq 1$). In the 11 cells where the no-defense ASR interval lies above $\sim 15\%$, the no-defense and EvoSafeHarness Wilson intervals are disjoint, so the reduction is not attributable to sampling noise. Pooling the discordant pairs across all fifteen cells, EvoSafeHarness neutralises 385 of the attacks that succeed undefended while opening 11 (a 35:1 ratio), a pooled McNemar $\chi^2_{cc} = 351.3$ ($p \approx 2 \times 10^{-78}$). The 11 newly-opened successes concentrate in the hardest weak-victim cells and correspond to the documented residuals (most of all DeepSeek/os-filesystem, the cell with the largest residual ASR in Table 1; see §7.3). Benign-utility Wilson intervals, by contrast, overlap between no-defense and EvoSafeHarness in all fifteen cells: with only 30 benign tasks per cell the modest per-cell utility cost is within sampling noise, consistent with the ~ 3 -point aggregate utility change reported in §6.1.