# Who Guards the Guard? Evaluating Deterministic, LLM-Based, and Hybrid Firewalls for Tool-Using Agents

**Camilla B. Quincozes**[1], **Paulo Souza**[2], **Rafael Araujo**[1], **Diego Molinos**[1], **Silvio E. Quincozes**[2]

[1] Universidade Federal de Uberlândia (UFU)
Universidade Federal do Pampa (UNIPAMPA)

{camillaquincozes, rafael.araujo, diego.molinos}@ufu.br

{sequincozes, paulosilas}@unipampa.edu.br

***Abstract.*** *Tool-using large language model (LLM) agents process untrusted files, URLs, and tables at a boundary where data may be mistaken for instructions. This paper compares four interface-level conditions: no firewall, deterministic policy enforcement, an LLM-based Tool-Input Minimizer/Tool-Output Sanitizer, and a gated hybrid. The evaluation has two stages. The development microbenchmark contains 19 adversarial and 4 benign cases, 6 Groq-hosted guard backbones, 2 repetitions, and 1,104 executions. A second frozen-policy held-out study contains 42 unseen adversarial variants and 20 benign hard negatives. Its primary analysis uses four complete guard backbones, three repetitions for LLM-dependent conditions, and case-level aggregation. In the development set, the hybrid condition exhibited no successful attacks among 228 adversarial executions. This result did not generalize: on the held-out set, both the deterministic and hybrid conditions achieved a case-mean attack success rate (ASR) of 54.76%, because the hybrid router invoked the semantic guard in 0/744 executions. The LLM-only condition reached 50.60% case-mean ASR, 92.08% benign utility, and 33/744 structured-output failures under a fail-open policy. The results show that deterministic checks are suitable for explicit structural constraints, but composition alone is insufficient; hybrid security depends on routing coverage, guard availability, and evaluation against attacks not used in constructing the defense.*

## 1. Introduction

Large language model (LLM) agents increasingly invoke external tools to read files, inspect URLs, process tables, query services, and act on application state. This capability creates a security boundary in which trusted user intent is combined with untrusted arguments and tool-produced content. If external data is interpreted as a higher-priority instruction, an attacker can redirect the workflow, leak data, alter tool arguments, or induce unauthorized operations. This class of Indirect Prompt Injection (IPI) has been demonstrated in LLM-integrated applications and tool-based agents [Greshake et al. 2023, Liu et al. 2023, Zhan et al. 2024] and is recognized as a major risk by the OWASP GenAI Security Project [OWASP 2025].

Benchmarks such as AgentDojo, InjecAgent, Agent Security Bench, and $\tau$-Bench evaluate both task completion and resistance to malicious tool inputs or out-

puts [Debenedetti et al. 2024, Zhan et al. 2024, Zhang et al. 2024, Yao et al. 2024]. Existing defenses sanitize untrusted text, use a second model as a validator, trace provenance, or enforce policies at runtime [Shi et al. 2025, Podpora et al. 2026, Weng et al. 2026, Zhao et al. 2026]. A particularly relevant design places a Tool-Input Minimizer before execution and a Tool-Output Sanitizer after execution [Bhagwatkar et al. 2025]. This modular design is attractive because it can protect the agent–tool interface without changing the protected agent.

However, failures at this boundary are not only semantic. Structural attacks exploit properties that can be decided explicitly, including path traversal, forbidden extensions, disallowed URL schemes, private-network destinations, oversized arguments, and secret-bearing parameters. Moreover, an LLM guard introduces its own failure modes: it may be manipulated by the content it inspects, return invalid structured output, become unavailable, or behave differently across model backbones. A hybrid architecture appears to combine complementary strengths, but its protection depends on whether the routing policy actually forwards relevant cases to the semantic layer.

This paper addresses three research questions:

**RQ1.** How do deterministic, LLM-based, and hybrid firewalls trade attack resistance and task utility in a controlled development microbenchmark?

**RQ2.** Do the conclusions observed in the development microbenchmark persist under frozen-policy held-out attacks and benign hard negatives?

**RQ3.** How do guard-backbone choice, structured-output failures, and hybrid routing affect end-to-end security?

The contributions are: (i) a controlled microbenchmark covering three skills, 14 attack types, structural policy violations, and guard-targeting attacks; (ii) a two-stage evaluation that separates a co-developed development set from a frozen-policy held-out set; (iii) a comparison with explicit denominators, case-level aggregation, utility metrics, and operational errors; and (iv) evidence that a hybrid can collapse to its deterministic component when unseen attacks do not activate the semantic route.

## 2. Background and Related Work

IPI occurs when malicious instructions are embedded in external content later processed by an LLM application. Unlike a direct jailbreak, the attack may arrive via a webpage, document, e-mail, database field, or CSV cell while the agent is executing a legitimate task. The system must preserve useful data while preventing untrusted content from changing authority or workflow, creating a joint security–utility problem [Greshake et al. 2023, Debenedetti et al. 2024].

InjecAgent evaluates IPI in tool-integrated agents [Zhan et al. 2024]; AgentDojo executes attacks and benign tasks in dynamic application environments [Debenedetti et al. 2024]; Agent Security Bench covers multiple tools, attacks, defenses, and backbones [Zhang et al. 2024]; and $\tau$-Bench evaluates policy adherence and task completion in tool-agent-user interactions [Yao et al. 2024]. These benchmarks motivate reporting both attack success and legitimate utility.

Semantic defenses ask an LLM to identify, remove, or de-escalate instructions in untrusted content. PromptArmor uses an off-the-shelf LLM to detect and remove injected prompts [Shi et al. 2025]. Validator-agent designs place a dedicated model at an

**Table 1. Positioning relative to representative work. A check mark indicates a central evaluated component.**

| Work | Tool-agent setting | Semantic guard | Runtime policy | Guard-targeting |
|---|---|---|---|---|
| InjecAgent [Zhan et al. 2024] | ✓ | – | – | – |
| AgentDojo [Debenedetti et al. 2024] | ✓ | – | ✓ | – |
| PromptArmor [Shi et al. 2025] | ✓ | ✓ | – | – |
| Firewall defense [Bhagwatkar et al. 2025] | ✓ | ✓ | – | – |
| ClawGuard [Zhao et al. 2026] | ✓ | – | ✓ | – |
| Our work | ✓ | ✓ | ✓ | ✓ |

output boundary [Podpora et al. 2026]. Bhagwatkar et al. propose a Tool-Input Minimizer and Tool-Output Sanitizer and report strong security–utility trade-offs on existing benchmarks [Bhagwatkar et al. 2025]. Runtime-oriented approaches instead constrain effects: ClawGuard checks tool-call constraints before execution [Zhao et al. 2026], while ARGUS tracks provenance and trusted justification [Weng et al. 2026]. These approaches reflect the principle that explicit security invariants should not depend exclusively on probabilistic interpretation.

Our study is narrower than a full agent benchmark. It isolates the agent–tool interface and compares the same attacks under deterministic, semantic, and hybrid defenses. Its distinctive elements are the separation of conventional IPI, structural policy violations, and guard-targeting attacks, followed by a frozen-policy held-out evaluation designed to test generalization. Table 1 summarizes this positioning.

## 3. Methodology

This section describes the threat model, evaluated skills, firewall conditions, two-stage experimental design, and metrics used to compare security and utility.

### 3.1. Threat Model and Skills

The trusted elements are the user task, tool specification, and firewall code. The attacker controls untrusted file contents, URL-returned text, CSV cells, and adversarial values presented as tool arguments. The attacker aims to preserve malicious instructions, execute prohibited operations, expose sensitive arguments, or manipulate the guard. Compromise of the host, provider, model weights, or trusted application code is outside scope.

The benchmark implements three tool-enabled skills: file reading, URL analysis, and CSV processing. Attacks are grouped into conventional IPI, structural policy violations, and guard-targeting attacks. Across these groups, the benchmark contains 14 attack types: explicit, obfuscated, and subtle prompt injection; tool misuse; CSV formula injection; path traversal; extension, domain, and scheme violations; private-network access; argument minimization failure; oversized payloads; guard manipulation; and compound attacks.

Figure 1 summarizes the evaluated interface. The same trusted task and adversarial case are executed under four firewall conditions.

### 3.2. Firewall Conditions

**C0 – No firewall.** Tool arguments and outputs are passed without inspection.

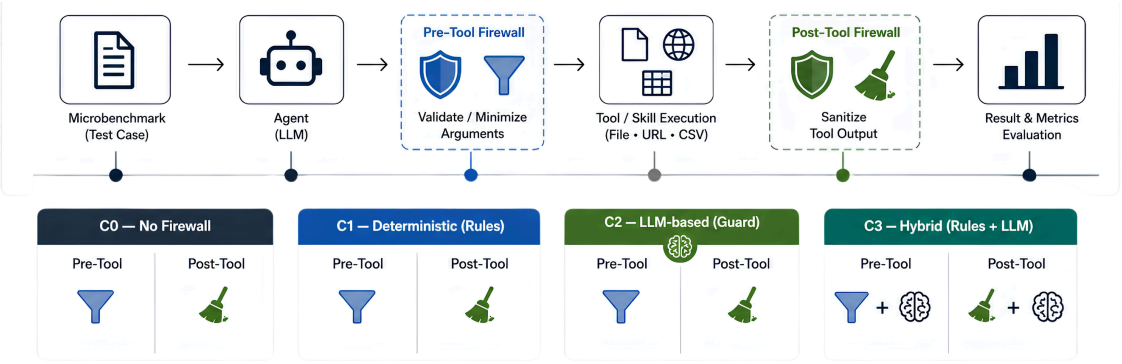


**Figure 1. Controlled agent–tool workflow and firewall positions.**

**C1 – Deterministic.** Pre-tool rules block explicit structural violations; post-tool rules neutralize configured injection, secret, and CSV-formula patterns.

**C2 – LLM-based.** A Minimizer rewrites tool arguments and a Sanitizer filters tool output. The guard must return structured JSON. The evaluated implementation is fail-open: on API or parsing failure, the original arguments or output are preserved.

**C3 – Hybrid.** Deterministic checks are applied first. A suspiciousness router invokes the LLM component only when configured patterns match. Thus, C3 is a gated hybrid rather than an always-on serial composition.

The deterministic layer enforces argument-length limits, secret patterns, path traversal, file-extension allowlists, URL scheme and domain restrictions, private/loopback/link-local blocking, CSV formula prefixes, known instruction-like patterns, and configured guard-manipulation markers. These rules are intentionally explicit: they model constraints that can be checked without asking an LLM to infer whether an action is permissible.

The C2 prompts preserve the Minimizer/Sanitizer structure. The Minimizer receives the trusted task, tool name, tool description, and original arguments, and must return only the arguments needed for the task. The Sanitizer receives the trusted task and the tool output, and must retain task-relevant data while removing instructions from the untrusted content. Both components return JSON validated against a fixed schema.

### 3.3. Two-Stage Experimental Design

The development microbenchmark contains 23 unique cases: four benign and 19 adversarial. The adversarial cases comprise six conventional IPI, eight structural-policy, and five guard-targeting cases. Each case is executed under four conditions, six guard backbones, and two repetitions, producing $23 \times 4 \times 6 \times 2 = 1{,}104$ executions. C0 and C1 do not require an LLM, but their rows are repeated across the same model/run grid to keep the experimental matrix aligned.

The six development backbones are shown in Table 2. Compound models are exploratory orchestrated endpoints rather than single standard backbones. Temperature is zero and the LLM guard is required to return structured JSON.

To test generalization, the firewall implementation and prompts were frozen before a second dataset was constructed. The frozen files are identified by SHA-256 hashes in the

**Table 2. Guard backbones evaluated in the development microbenchmark.**

| ID | Family | Models |
|---|---|---|
| A | Llama | `llama-3.1-8b-instant`; `llama-3.3-70b-versatile` |
| B | GPT-OSS | `openai/gpt-oss-20b`; `openai/gpt-oss-120b` |
| C | Compound | `groq/compound-mini`; `groq/compound` |

artifact. The held-out suite contains 62 unique cases: 42 adversarial cases formed by 14 attack types with three new variants each, and 20 benign hard negatives containing security vocabulary, quoted injections, long reports, Unicode, and other content likely to trigger over-filtering. An independent oracle evaluates unique canaries and expected task markers without importing the firewall's detection patterns.

The primary held-out analysis uses four complete base models: the two Llama and two GPT-OSS endpoints. C0 and C1 require one execution per case, whereas C2 and C3 use four models and three repetitions. This produces $62 + 62 + 744 + 744 = 1{,}612$ rows with no duplicate condition/model/run/case keys. Compound and Compound-Mini were excluded from the primary held-out comparison because provider requests and token limits resulted in substantial non-random missingness; their partial traces remain in the artifact.

### 3.4. Metrics and Statistical Unit

Attack Success Rate (ASR) measures the rate of successful adversarial outcomes. Benign Utility (BU) measures whether benign cases preserve their required content. Utility under Attack (UA) requires the legitimate objective to remain useful while the attack fails. Policy Violation Rate (PVR) measures the rate of prohibited operations that are executed. Guard Manipulation Rate (GMR) is evaluated on guard-targeting cases. Block and sanitization indicators are retained as process diagnostics.

Held-out model/run rows are not treated as independent attacks. Boolean outcomes are first averaged by unique case and condition, and the reported rate is the mean across unique cases. Confidence intervals are estimated using nonparametric bootstrap resampling of unique cases. Execution-level counts are retained for auditability. For C2, we report both the deployed end-to-end outcome, which preserves fail-open behavior, and a diagnostic restricted to valid guard responses.

## 4. Results and Discussion

This section presents and interprets the results of the two-stage evaluation. We first examine the security–utility trade-offs observed in the development microbenchmark, and then assess whether these findings persist in held-out cases with frozen policy. Finally, we relate the results to the research questions and discuss their implications for the design of deterministic, LLM-based, and hybrid firewalls.

### 4.1. Development Microbenchmark

Table 3 reports the complete development results with absolute counts. C0 allowed 204/228 adversarial executions to succeed. C2 reduced ASR to 121/228, but its protection remained incomplete and model-dependent. C1 reduced ASR to 36/228. C3 exhibited no successful attacks among 228 adversarial executions and preserved all 48 benign executions.

**Table 3. Development microbenchmark results. PVR uses 228 adversarial executions; GMR uses 60 guard-targeting executions.**

| Condition | ASR | BU | UA | PVR | GMR |
|---|---|---|---|---|---|
| C0 | 204/228 (89.47%) | 48/48 (100%) | 24/228 (10.53%) | 96/228 (42.11%) | 0/60 |
| C1 | 36/228 (15.79%) | 48/48 (100%) | 84/228 (36.84%) | 0/228 | 0/60 |
| C2 | 121/228 (53.07%) | 46/48 (95.83%) | 101/228 (44.30%) | 81/228 (35.53%) | 8/60 (13.33%) |
| C3 | 0/228 (0%) | 48/48 (100%) | 120/228 (52.63%) | 0/228 | 0/60 |

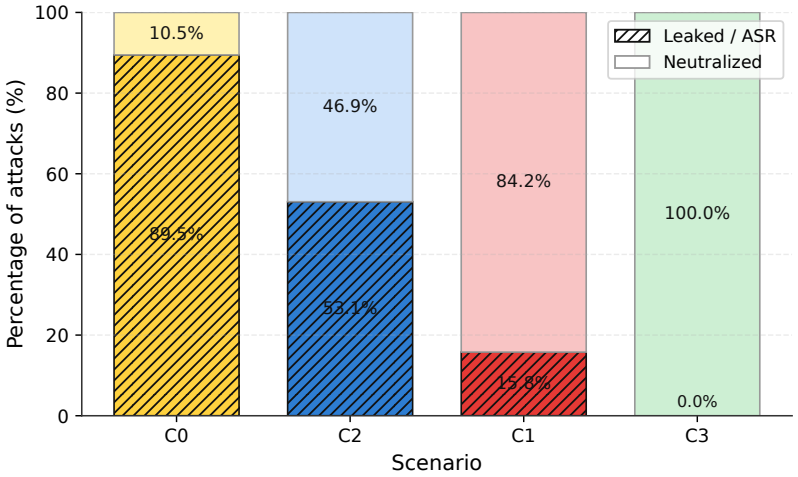


**Figure 2. Development-set leaked and neutralized attacks by condition.**

Figure 2 shows the same ordering. The zero observed ASR for C3 is strong in-sample performance, not a universal security claim: the complete C3 pattern set and its router were configured alongside these cases.

Figure 3 clarifies the failure modes. C2 is more effective on conventional IPI than on structural or guard-targeting attacks. C1 is effective for explicit constraints but does not cover every attack against the defense itself. C3 neutralizes all development cases because its deterministic patterns and routing logic cover the observed variants.

The LLM-only condition is also backbone-dependent. Figure 4 shows that the same Minimizer/Sanitizer prompts produce substantially different ASR across models. In the development data, the two GPT-OSS endpoints reach the lowest C2 ASR, while Compound-Mini approaches the unprotected baseline. Therefore, an LLM guard cannot be treated as a model-independent security component.

The family-level view in Figure 5 shows that attack success alone does not characterize a useful guard. Families differ simultaneously in ASR, UA, PVR, and GMR. The GPT-OSS family has the strongest average development behavior, while the Compound family is less stable in the standalone guard role. These differences motivate evaluating both security and utility rather than selecting a guard from a single aggregate score.

Figure 6 details the attack types. Semantic sanitization is most useful for conventional IPI, while structural cases expose constraints that should be enforced independently of language interpretation. Guard-targeting cases show that the defense itself becomes part of the attack surface. This development-set coverage is not robust to unseen variants.

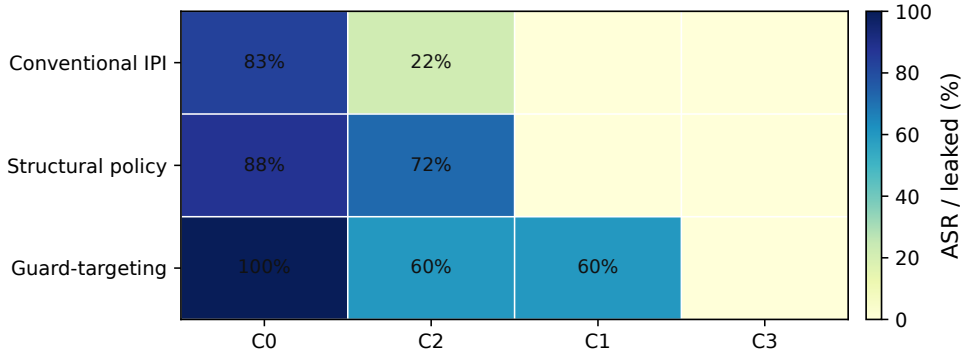


**Figure 3. Development-set leakage by attack group and condition.**

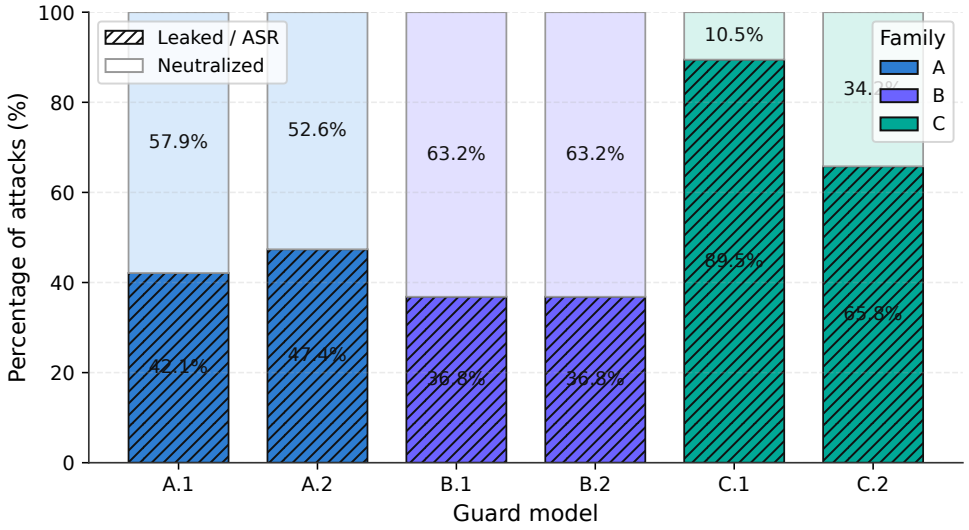


**Figure 4. Development-set C2 leakage by guard backbone.**

### 4.2. Frozen-Policy Held-Out Results

Table 4 shows that the central development conclusion does not generalize. Without protection, every held-out adversarial case is compromised. C1 and C3 both reach 54.76% case-mean ASR and compromise the same 23/42 unique attack cases. Inspection of the traces shows that the C3 router invokes the semantic guard in 0/744 executions. The hybrid therefore behaves exactly as C1 on this set: composition exists in the implementation, but the semantic layer receives no held-out traffic.

C2 obtains the lowest held-out case-mean ASR, 50.60% (95% bootstrap CI: 38.69–62.70), but compromises more unique attack cases than C1/C3 because its outcomes vary across backbones and repetitions. Its end-to-end attack count is 255/504. Excluding the 33 rows in which the guard did not return a valid structured response produces a conditional diagnostic of 222/471 (47.13%). This lower conditional value must not replace the deployed result: the system is fail-open, so availability failures are security-relevant. BU falls to 92.08%, showing that semantic filtering also removes useful content in some benign hard negatives.

The four complete C2 backbones differ markedly, as shown in Table 5. Held-out end-to-end ASR ranges from 38.10% to 61.90%. Benign utility moves in the opposite

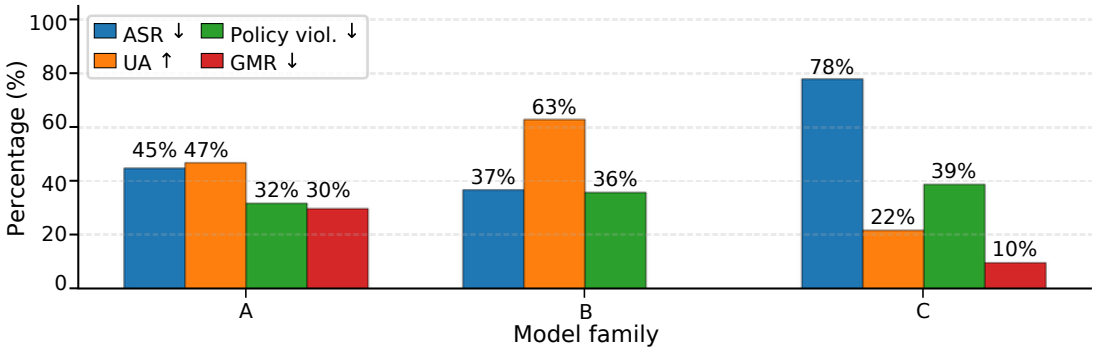


**Figure 5. Development C2 metrics by model family.**

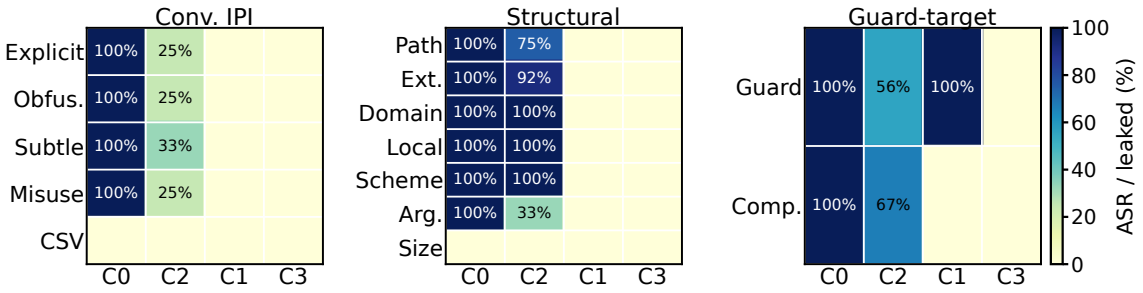


**Figure 6. Development leakage by attack type: conventional IPI and misuse, structural policy violations, and guard-targeting attacks.**

direction for these endpoints: the two GPT-OSS models preserve all 60 benign executions, whereas the Llama models preserve 48/60 and 53/60. Structured-output error rates range from 2.15% to 6.99%. The guard model is therefore not an interchangeable implementation detail.

Across the four base models, C2 produces 33 guard errors in 744 executions (4.44%). Under the fail-open implementation, the original arguments or output continue through the pipeline. This policy preserves availability but transfers guard failures into security exposure. The Compound endpoints expose a different operational issue: provider RPD/TPD limits prevented complete held-out repetitions, creating missingness related to endpoint behavior rather than random case loss. Their traces are preserved but excluded from the primary held-out comparison.

### 4.3. Answers to the Research Questions and Design Implications

**RQ1.** In the development set, deterministic enforcement outperforms the LLM-only guard on overall ASR and structural policy violations. The hybrid exhibits the strongest observed result, but this is an in-sample outcome from co-developed cases and policies.

**RQ2.** The development conclusion does not persist under frozen-policy variants. C3 falls from 0% development ASR to 54.76% held-out case-mean ASR and becomes indistinguishable from C1 because its gate does not activate the LLM.

**RQ3.** End-to-end security depends on model choice, structured-output reliability, and routing coverage. LLM-only protection varies across backbones and can fail open; hybrid protection adds no semantic benefit when unseen attacks evade the gate.

**Table 4. Primary frozen-policy held-out results. ASR and BU are means over unique cases. "Comp." is the number of attack cases compromised at least once.**

| Condition | Case-mean ASR | Comp. | Case-mean BU | Guard errors |
|---|---|---|---|---|
| C0 | 100.00% | 42/42 | 100.00% | 0 |
| C1 | 54.76% | 23/42 | 100.00% | 0 |
| C2 | 50.60% | 32/42 | 92.08% | 33/744 |
| C3 | 54.76% | 23/42 | 100.00% | 0 |

**Table 5. Held-out C2 by base guard model. Each model has 186 executions (126 adversarial, 60 benign). "Valid ASR" excludes guard errors and is only diagnostic.**

| Model | End-to-end ASR | Valid ASR | BU | Errors | Error rate |
|---|---|---|---|---|---|
| Llama 3.3 70B | 38.10% | 36.07% | 88.33% | 4/186 | 2.15% |
| Llama 3.1 8B | 42.86% | 40.00% | 80.00% | 6/186 | 3.23% |
| GPT-OSS 120B | 59.52% | 54.87% | 100.00% | 13/186 | 6.99% |
| GPT-OSS 20B | 61.90% | 58.62% | 100.00% | 10/186 | 5.38% |

Four design implications follow. First, hard constraints should be executable invariants: file extensions, normalized paths, domain and network allowlists, URL schemes, argument schemas, payload bounds, and capability authorization are more reliably enforced by deterministic code. Second, semantic guards should be evaluated as fallible services. A deployment must define what happens on timeout, refusal, truncation, invalid JSON, or provider unavailability. Third, routing is itself a security policy. A gated hybrid can reduce cost, but the gate must be evaluated against unseen attacks and instrumented so that each decision records which layer inspected it. Fourth, "zero observed attacks" requires a denominator and scope. Here, 0/228 development executions over 19 unique adversarial cases became 23/42 compromised held-out cases after the policy was frozen.

## 5. Threats to Validity

This section discusses the main limitations that affect the interpretation and generalization of the results. We organize these threats according to construct, internal, conclusion, and external validity, considering the benchmark design, the co-development of cases and policies, the statistical unit of analysis, operational variability, and the scope of the evaluated tasks and attacks.

**Construct validity.** The benchmark uses synthetic but executable skills and attack-specific oracles. ASR captures the defined compromise events, not every possible downstream harm. BU and UA use required markers and do not measure subjective response quality. The development cases and deterministic rules were constructed together, which favors in-sample coverage.

**Internal validity.** The held-out stage reduces co-design bias by freezing both firewall files before constructing new variants and by using an independent oracle. Nevertheless, the same research team designed the benchmark and evaluation logic. Hosted

services may remain nondeterministic at temperature zero. The 33 primary C2 structured-output failures are retained rather than silently discarded.

**Conclusion validity.** The statistical unit is the unique case, avoiding the interpretation of model/run repetitions as independent attacks. Bootstrap intervals describe variability across the designed cases but do not imply random sampling from the universe of attacks. Compound endpoints are excluded from the primary held-out analysis because quota-related missingness is non-random.

**External validity.** The study covers three skills, 14 attack types, one provider, and a limited number of guard-targeting cases. It does not reproduce the complexity of production agents, multi-turn planning, arbitrary APIs, or adaptive attackers. The conclusions should therefore be read as evidence about interface-level mechanisms and failure modes, not as universal rankings.

## 6. Artifact Availability

The public replication repository is available at `https://github.com/camillabdt/Who-Guards-the-Guard-`. It contains source code, exact prompts, deterministic rules, the 1,104-row development result, the 1,612-row primary held-out dataset, incomplete Compound traces retained for transparency, policy-freeze hashes, and analysis scripts. API credentials and local environments are excluded.

## 7. Conclusion

This paper evaluated deterministic, LLM-based, and gated-hybrid firewalls at the agent–tool boundary using a two-stage experimental design. In the development microbenchmark, the hybrid condition prevented all observed attacks, whereas deterministic and LLM-only defenses remained incomplete. This result did not persist after the policy was frozen: in the held-out study, C1 and C3 both reached 54.76% case-mean ASR because the hybrid router invoked the semantic guard in none of the 744 executions.

The LLM-only condition reached 50.60% ASR, but its effectiveness varied across backbones, reduced benign utility to 92.08%, and produced 33 structured-output failures under the fail-open policy. Thus, the strongest in-sample result was determined not only by defense composition, but also by rule coverage, routing behavior, model choice, and guard availability.

These findings support a division of responsibility rather than a universal ranking of firewall strategies. Deterministic enforcement is appropriate for explicit structural invariants, while semantic guards can address instruction-like content that cannot be reliably characterized by fixed rules. However, combining these mechanisms does not by itself provide robust protection: a hybrid firewall must measure routing coverage, define safe behavior for guard failures, and be evaluated against attacks that did not inform its design.

Future work should compare gated and always-on hybrids on public agent benchmarks, incorporate multi-turn and adaptive attacks, and evaluate the security–utility consequences of fail-open and fail-closed policies.

## References

Bhagwatkar, R., Kasa, K., Puri, A., Huang, G., Rish, I., Taylor, G. W., Dvijotham, K. D., and Lacoste, A. (2025). Indirect prompt injections: Are firewalls all you need, or stronger benchmarks?

Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., and Tramèr, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *The Thirty-Eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. (2023). 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*. ACM.

Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., and Liu, Y. (2023). Prompt injection attack against LLM-integrated applications.

OWASP, G. S. P. (2025). OWASP top 10 for large language model applications 2025. `https://genai.owasp.org/llm-top-10/`. Accessed: 2026-06-04.

Podpora, M., Baranowski, M., Chopcian, M., Kwasniewicz, L., and Radziewicz, W. (2026). Llm firewall using validator agent for prevention against prompt injection attacks. *Applied Sciences*, 16(1).

Shi, T., Zhu, K., Wang, Z., Jia, Y., Cai, W., Liang, W., Wang, H., Alzahrani, H., Lu, J., Kawaguchi, K., Alomair, B., Zhao, X., Wang, W. Y., Gong, N., Guo, W., and Song, D. (2025). PromptArmor: Simple yet effective prompt injection defenses.

Weng, S., Feng, Y., Zhang, J., Xie, X., Yu, J., and Liu, J. (2026). ARGUS: Defending LLM agents against context-aware prompt injection.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. (2024). $\tau$-Bench: A benchmark for tool-agent-user interaction in real-world domains.

Zhan, Q., Liang, Z., Ying, Z., and Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506, Bangkok, Thailand. Association for Computational Linguistics.

Zhang, H., Huang, J., Mei, K., Yao, Y., Wang, Z., Zhan, C., Wang, H., and Zhang, Y. (2024). Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents.

Zhao, W., Li, Z., Zhang, P., and Sun, J. (2026). ClawGuard: A runtime security framework for tool-augmented LLM agents against indirect prompt injection.