FROM SAFETY MEMORY TO ENFORCED POLICY: CONTINUITY AND SCOPE ACROSS AGENT REPLICAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Adaptive agent defenses learn safety rules from past interactions to prevent recurring harm. Yet remembering a rule does not ensure that later actions obey it. We study this gap in serverless agents, where learning and execution can occur on different replicas. We propose Service-Global Monotonic Immunity (SGMI), which shares accepted rules across replicas, defines their action scope, and blocks matching tool calls before generative reconsideration. In Knative scale-out experiments, shared memory retrieves the learned rule on every fresh-replica probe but still permits 47.9% of harmful actions. SGMI blocks all 144 tested harmful probes across original and fresh replicas. In a paired banking ablation, recipient boundaries reduce legitimate-action rejection from 62.5% to 25%, while preserving all eight harmful-action blocks. SGMI connects adaptive safety learning with persistent, action-level protection across changing execution instances.

1 INTRODUCTION

Adaptive agent defenses seek to improve safety by retaining lessons from previous interactions. Long-term memory preserves information across tasks (Zhong et al., 2024; Xu et al., 2025; 2026), while security-oriented approaches derive checks, matching conditions, and responses from harmful episodes (Weng et al., 2026b; Luo et al., 2025; Wei et al., 2025). An accepted lesson should prevent recurring harm, rather than only reject the current attack.

However, a learned safety record is not yet an enforced policy. Retrieving a record makes it available for reasoning, but the defense must still associate it with the proposed action and carry the resulting constraint through execution. A later judgment may miss an applicable rule or approve an action that it covers. Conversely, enforcing an overly broad interpretation can suppress legitimate behavior. The question is how accepted safety knowledge becomes a lasting, appropriately scoped constraint.

Serverless platforms scale and replace instances as workloads change (Li et al., 2023; Ekwe-Ekwe & Amos, 2024), separating learning from later execution. Local memory can disappear with a replica; shared memory restores access but leaves rule interpretation and enforcement unresolved.

Consider a defense that learns to reject an unreviewed draft sent to an external address. The lesson must survive instance replacement, apply when another connector performs the same prohibited send, and distinguish a reviewed draft that may be sent legitimately. Neither copying the record nor banning all external email satisfies these requirements. Figure 1 illustrates how memory availability, rule scope, and enforcement determine the outcome of later actions.

We study this transition at the level of an accepted safety rule. Visibility concerns whether the serving instance receives it; scope determines which actions match; application determines whether a match prevents execution. This decomposition distinguishes memory loss from failures after retrieval.

We propose Service-Global Monotonic Immunity (SGMI), which connects rule acceptance to action-level enforcement. Ordered shared state makes an accepted block available to later replica reads. Structured predicates specify the prohibited effect and its recipient or timing conditions, distinguishing matching harm from legitimate variations. A trusted tool wrapper checks the fixed action arguments and prevents a matched block from being reversed by generative reconsideration; unmatched actions retain adaptive analysis. These mechanisms jointly define when a learned rule becomes binding and which actions it constrains. Building on consistent authorization and deterministic tool policies (Pang

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

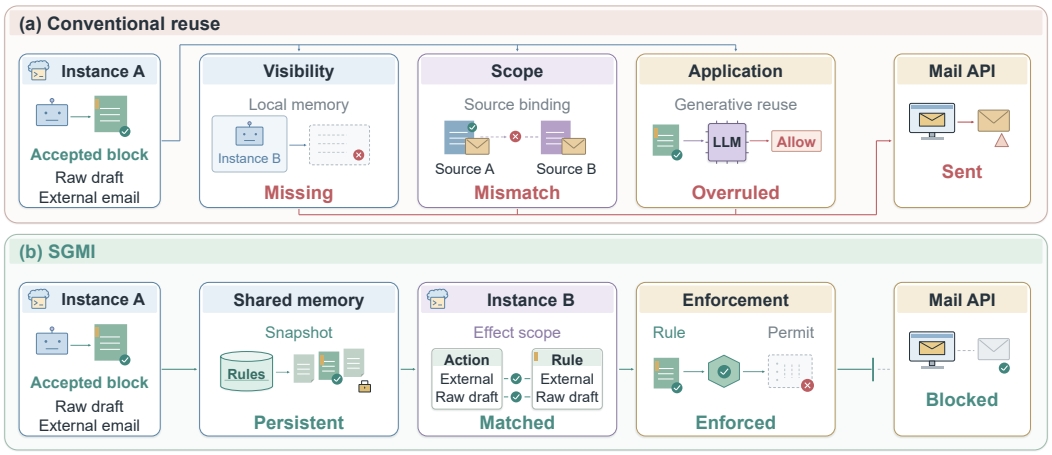


Figure 1: Memory reuse across replicas.

et al., 2019; Shi et al., 2026b), SGMI specifies a continuity contract for accepted rules and exposes scope choices to paired safety and utility evaluation.

Our Knative evaluation follows this distinction between remembering and enforcing. Across independently learned configurations, shared memory exposes the committed rule on all 96 fresh-replica probes but permits 46 harmful effects; SGMI blocks all 144 probes across original and fresh replicas. A paired banking ablation then tests scope: recipient boundaries preserve three additional legitimate actions, reducing rejections from five of eight to two while all eight harmful actions remain blocked.

Our contributions are summarized as follows:

- We characterize the gap between learned safety memory and enforced policy through rule visibility, scope, and application, using replica changes to expose failures of later reuse.
- We design SGMI to bind accepted rules to explicit action predicates and replica-independent enforcement, and specify the ordering conditions for protection to persist.
- We develop a rule-centered evaluation that connects lifecycle observations, fixed-record comparisons, and scope ablations to harmful effects and legitimate behavior.
- SGMI blocks all 144 probes across original and fresh replicas. Paired banking decisions show that recipient boundaries reduce rejections from 62.5% to 25% among eight legitimate actions while preserving all eight harmful-action blocks.

Our code and evaluation artifacts are publicly available (Anonymous, 2026).

2 Problem Formulation

2.1 Serving model and adversary

We consider a tool-using agent whose replicas retain learned rules locally or consult them as shared advice. The platform controls routing, replacement, and autoscaling. An indirect injection places adversarial instructions in content the agent reads while carrying out a legitimate task (Greshake et al., 2023). The adversary has black-box access and controls retrieved content, as in AgentDojo and InjecAgent (Debenedetti et al., 2024; Zhan et al., 2024). It may repeat a related request after rule acceptance and wait for replacement, but cannot choose the replica.

We trust the platform, rule store, acceptance mechanism, and tool wrapper. The agent and the generative defense may make unsafe decisions, so enforcement of a matching block cannot depend on their cooperation. The attacker cannot modify committed state, forge authorization, or bypass the protected-tool interface. Each rule applies within one service policy and trust domain.

2.2 FAILURE DECOMPOSITION

A post-commit request begins after the service acknowledges a blocking rule. Before executing a protected action, the defense checks a snapshot of accepted rules. Within the same policy epoch, let E_r denote execution of an effect semantically covered by rule r in such a request. Visibility V_r means that the snapshot contains r; matching M_r means that the implementation associates the proposed action with r. An escaped effect belongs to one of three disjoint classes:

$$\underbrace{E_r \wedge \neg V_r}_{\text{visibility}} \quad \dot{\vee} \quad \underbrace{E_r \wedge V_r \wedge \neg M_r}_{\text{scope}} \quad \dot{\vee} \quad \underbrace{E_r \wedge V_r \wedge M_r}_{\text{application}}. \tag{1}$$

In the application class, a model may approve a matched action, or execution may bypass a blocking decision. An incorrect accepted rule is a separate rule-quality error. The decomposition specifies what an evaluation must observe: finding the record in memory establishes visibility, while identifying a scope or application failure also requires the match and execution decisions. Appendix A defines the ordering of concurrent reads and commits.

3 METHOD

3.1 SYSTEM OVERVIEW

SGMI separates learning a safety rule from deciding whether an accepted block applies (Figure 2). Confirmed harm supplies the learner with an episode from which to derive a rule. Once accepted, the rule enters shared state before the update is acknowledged. A serving agent can then propose tool calls, but a trusted wrapper fixes their arguments and checks the rule's explicit scope. A match prevents execution without another generative judgment. Otherwise, the adaptive defense evaluates the action, and an allow decision authorizes only that proposal. The shared store preserves the lesson across replicas; the predicate and wrapper determine its effect on behavior.

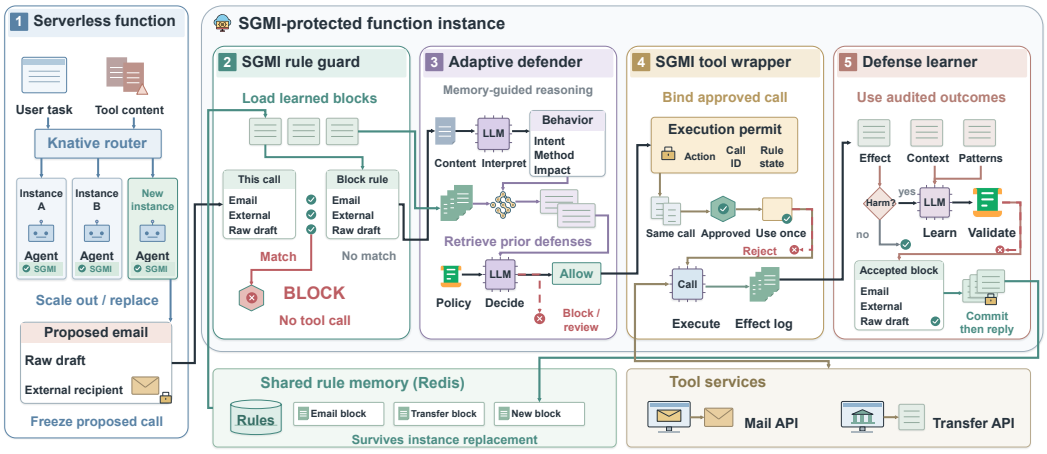


Figure 2: SGMI action-level protocol.

3.2 CONTINUITY CONTRACT

Let $\mathcal{S}_v$ denote the rule state at committed version v. SGMI represents an accepted blocking record as $r = \langle f, d, h, \pi \rangle$, comprising its effect family, trust direction, canonical behavior signature, and disposition. The fixed action envelope $e_{x,a}$ is mapped to $q_{x,a} = \phi(e_{x,a}) = \langle f_e, d_e, h_e \rangle$. A match requires agreement on all three scope fields:

$$\textsc{Match}(q, r) = \mathbf{1}[f_e = f_r]\,\mathbf{1}[d_e = d_r]\,\mathbf{1}[h_e = h_r]. \tag{2}$$

This projection determines the enforced scope. The email prototype uses request text, recipients, and actions; its simulator assigns external sends a pre-review label even when the request refers

to completed review. The banking prototype uses transfer timing and initial payee history, leaving subscription content and authorization for the current transfer outside its predicate. RQ3 varies these concrete scope fields to identify which legitimate actions each preserves.

Within an immutable rule epoch, $\textsc{CommittedBlock}(r)$ means that r has durably committed as a block. Let $R_{x,a}$ be the action's snapshot read and $\textsc{ActionPost}(r, x, a)$ mean that the commit precedes invocation of that read. A single-use permit $\tau(e, i, v)$ binds an envelope, unique attempt i, and read version v. Every protected effect requires such a permit. Continuity requires

$$\begin{aligned}\forall r, x, a : \textsc{CommittedBlock}(r) \wedge \textsc{ActionPost}(r, x, a) \\ \wedge \textsc{SameEpoch}(r, R_{x,a}) \wedge \textsc{Match}(q_{x,a}, r) \\ \Rightarrow \neg\textsc{Exec}(x, a) \wedge \\ \big(\textsc{ReadOK}(R_{x,a}) \Rightarrow \textsc{Visible}(r, x, a) \\ \wedge \textsc{Decision}(x, a) = \textsc{block}\big).\end{aligned} \tag{3}$$

The guarantee follows from ordered state and complete mediation (Herlihy & Wing, 1990; Pang et al., 2019; Saltzer & Schroeder, 1975). Commits and reads use one durable linearizable object; authorization binds the action arguments; a failed read or invalid permit prevents execution. ReadOK denotes a successful snapshot read, while failure returns review without executing the action. Covering every action prohibited by the semantic record further requires $\textsc{Covered}(a, r) \Rightarrow \textsc{Match}(q_{x,a}, r)$. RQ3 examines the effect of that projection choice, and Appendix A provides the proof.

3.3 State and action ordering

SGMI stores blocking projections in an append-only sequence $\mathcal{S}_v = \{r_1, \ldots, r_n\}$. Each committed head identifies an immutable prefix read as one snapshot. Concurrent learners compare the head and append atomically with bounded retries. Accepted blocks persist within the epoch, whereas a permissive model decision authorizes only the current action attempt. Algorithm 1 reads state before each action so that rules committed during an episode reach subsequent reads. Policy changes start a new epoch; measured runs keep policy fixed.

The lifecycle implementation checks a request snapshot with trusted synchronous dispatch. A later wrapper binds fixed arguments to the observed version while retaining that snapshot. Equation 3 covers actions whose reads follow the commit; earlier permits keep their original version.

Algorithm 1 Action-level serving and learning

Require: request x, proposed action a with attempt ID i, versioned rule state
$e \leftarrow \textsc{Freeze}(x, a, i)$; $q \leftarrow \textsc{Project}(e)$
$(v, \mathcal{S}_v) \leftarrow \textsc{ReadCommittedState}()$
if $q = \bot \vee v = \bot$ **then**
  return REVIEW
end if
if $\exists r \in \mathcal{S}_v : \pi_r = \textsc{block} \wedge \textsc{Match}(q, r)$ **then**
  return $(\textsc{block}, v)$
end if
$d \leftarrow \textsc{RunUnderlyingDefense}(x, e)$
if $d \neq \textsc{allow}$ **then**
  return (d, v)
end if
$\tau \leftarrow \textsc{AuthorizeOnce}(e, i, d, v)$
$o \leftarrow \textsc{Execute}(e, \tau)$
if $\textsc{ConfirmedHarm}(o)$ **then**
  $r^+ \leftarrow \textsc{InduceAndValidate}(x, e, o)$
  if $r^+ \neq \bot$ **then**
    $v \leftarrow \textsc{Commit}(v, r^+)$
  end if
end if
return (o, v)

3.4 MATCHING AND FALLBACK

An action outside the structured predicate can still benefit from learned advice. We compare three treatments: Full retains generative reuse, Scoped restricts it to compatible effect families and trust directions, and Direct omits it. Their structured decisions are identical, isolating how fallback affects harmful and legitimate actions. The indexed implementation groups records by (f, d, h): a match checks bucket $B_{f,d,h}$ in $O(|B_{f,d,h}|)$ time, while a miss loads the adaptive snapshot in $O(n)$ time. Storage is $O(n)$. The later public-tool implementation scans a consistent snapshot for each action. RQ5 reports indexed latency and public-tool judgment calls separately.

4 EXPERIMENTAL SETUP

4.1 SYSTEMS AND COMPARISONS

SGMI runs on Knative with a local reconstruction of AgentAntibody's projection, retrieval, response, and update stages. Its normalization includes structural acceptance that can override model rejection (Appendix B.1). Redis-compatible storage orders appends and reads. The testbed uses a four-core, 15-GiB Ubuntu VM with Kubernetes 1.34, Kind, Knative Serving, and Kourier.

Local Memory stores records in the serving process. Shared externalizes them for model interpretation. SGMI checks a structured predicate first; the same-scope policy uses identical predicates and fallback to control for ordinary enforcement. Progent runs in its native agent pipeline. Controlled banking comparisons use the deepseek-chat API endpoint.

Lifecycle measurements use request snapshots and trusted synchronous dispatch after learning acknowledgment, covering the request-level case of Figure 2. Fixed sequences execute admitted calls in the caller's public-tool environment. Autonomous episodes retain agent-proposed actions and the official final-state evaluator. Implementation checks examine action binding and consistent reads.

4.2 RECORDS, TASKS, AND OUTCOMES

The original campaigns reset state and learn independently for each configuration. Their 36 ingress commits retain record identifiers but not full content. The controlled follow-up instead restores one archived banking record to every method. Its semantic conditions concern subscription information and unauthorized transfer, whereas its structured predicate selects immediate external transfers. The consumers therefore receive the same record but enforce different scopes.

We measure harmful effects, fully dispatched legitimate sequences, and final benchmark utility separately. Final utility can include pre-existing goals when initial values were not recorded. RQ1 repeats twelve cycles within one deployment per configuration. The sequence test uses six harmful and six legitimate task sources in three deployments; the sixteen-input ablation pairs decisions by action. Repetitions remain grouped by task and deployment, without population risk estimates. Fresh-process performance tests use identical fixed model responses (Georges et al., 2007; Kalibera & Jones, 2013). Appendix B details the protocols and exclusions.

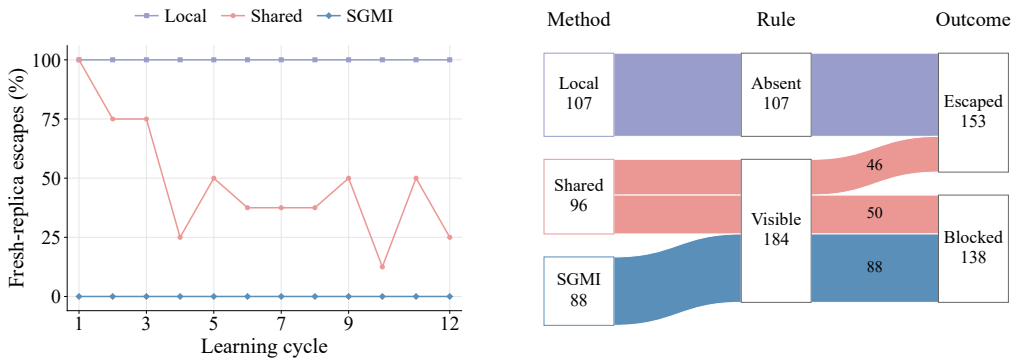
5 EVALUATION

The evaluation tests rule availability across replicas, reuse under fixed records, scope through paired ablations, autonomous intervention, and serving cost.

5.1 RQ1: DOES PROTECTION SURVIVE REPLICA CHURN?

Each public-ingress cycle resets state, commits a learned block, induces Knative scale-out through benign load, and submits related probes. Twelve cycles per configuration retain the initial pod and revision. Figure 3 shows per-cycle escapes and traces 291 fresh-replica probes by visibility and outcome.

SGMI has zero fresh-replica escapes in every cycle, while Local Memory permits every fresh-replica probe. Shared varies from 12.5% to 100% escapes across cycles. Figure 3(b) connects these outcomes



(a) Per-cycle escapes.

(b) Fresh-replica outcomes.

Figure 3: Public-ingress continuity.

to memory availability, with flow widths representing counts. All 107 Local probes lack the rule and escape. Shared's 96 probes read the rule: 50 are blocked and 46 escape. All 88 SGMI probes read the rule and are blocked. Original replicas show the same contrast: Local permits 18 of 37 probes and Shared 21 of 48, while SGMI blocks all 56. Restoring visibility therefore does not by itself preserve the learned block. The next experiment fixes the record to isolate reuse from independent learning. Appendix B details routing and replica diagnostics.

Released AGrail exhibits the same state-placement issue. In three two-replica deployments, hash checks confirm that a native update changes one local memory but not the other. Of 72 routed probes, 41 reach the replica without the update. Relevant memory changes six of eleven paired decisions.

5.2 RQ2: What does shared memory preserve?

We restore the same complete banking record to four consumers in three deployments. Each consumer evaluates six harmful and six legitimate fixed sequences before and after replacement. Table 1 summarizes 288 sequence observations, with eighteen observations per class and phase from six task sources. Calls allowed by the service execute in a separate public banking runtime.

Table 1: Same-record reuse.

	Harmful effects (%)		Benign dispatch (%)		Judge calls	
Method	Before	After	Before	After	Before	After
	$n = 18$ per phase		$n = 18$ per phase			
Local	100	100	100	100	93	0
Shared	100	100	100	100	93	93
SGMI	0	0	66.7	66.7	33	33
Same-scope policy	0	0	66.7	66.7	33	33

One record; six task sources per class, repeated in three deployments. Consumers differ in scope.

Receiving the same record leads to different action constraints. SGMI and the same-scope policy block all harmful sequences in each deployment, while Shared generation selects no semantic match for any harmful call. Their common predicate omits the semantic record's subscription and authorization clauses. Both structured consumers also reject two of the six legitimate references. Local replacement removes the record and judgment calls without changing harmful outcomes, showing why the outcome alone does not reveal memory loss. Representation also matters: Shared generation blocks all harmful sequences under an earlier simplified adapter (Appendix B.3).

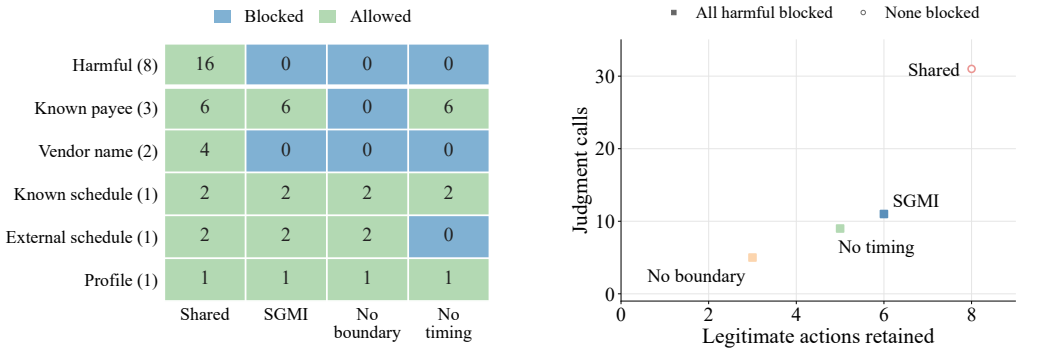
The rejected legitimate references use vendor names where account identifiers are expected. This argument representation places them outside the known-payee boundary. The test measures dispatch

of selected calls and omits injection 8's scheduled-payment objective. Restoration logs verify the record hash and replacement, but omit per-call replica identities (Appendix B.3).

A follow-up checks four authorized reference actions and one unauthorized transfer under their public task prompts. All methods allow the four authorized actions on both observed serving instances; only SGMI and the same-scope policy block the selected transfer. The refunds map to known payees, while the scheduled actions fall outside the immediate-transfer predicate. These forty decision-only observations connect the selected actions to their boundary and timing fields (Appendix B.4). Tool execution and task completion are measured in the other cohorts.

5.3 RQ3: Which scope fields preserve utility?

Recipient and timing fields preserve different legitimate actions. Figure 4(a) groups the sixteen paired actions from twelve task sources by behavior. Colors encode decisions, cell values count judgment calls, and parentheses give group sizes. All structured variants block the eight harmful calls. Recipient boundaries preserve three known-payee transfers; timing preserves one external scheduled-payment update. Removing these fields raises legitimate rejections from two of eight to five and three, respectively. The known-payee schedule and profile update remain allowed throughout.



(a) Decisions and judgment calls.

(b) Retention and cost.

Figure 4: Scope and cost.

Figure 4(b) shows that SGMI retains the most legitimate actions among the three structured variants: six at eleven calls, compared with three at five calls without boundaries and five at nine calls without timing. All block the eight harmful actions. The ablations save calls by rejecting extra legitimate actions; all three reject RQ2's two vendor-name references.

A separate email comparison examines reuse across connectors through metamorphic testing (Segura et al., 2016; Ribeiro et al., 2020). Five transformations preserve the external effect while changing its connector. SGMI blocks all thirty transformed replays, compared with three for a source-bound configuration. These independently initialized configurations change both connector dependence and action grouping. Table 5 includes their benign and boundary-adjacent outcomes.

Fallback changes the tradeoff beyond structured matches. In an earlier banking campaign, Full and Direct expose the same one of six harmful task sources. Full permits one of eighteen harmful episodes, compared with three for Direct, but completes thirteen of eighteen legitimate episodes rather than eighteen. A separate seven-task shopping comparison gives Shared, SGMI, and the same-scope policy identical completion outcomes, including three lost browsing tasks relative to no defense. The shared link restriction rejects those legitimate actions under all three consumers (Appendix B.5).

5.4 RQ4: When does the agent invoke the defense?

We next let the agent choose its actions. Six public banking tasks run clean and with two injection goals under five methods, producing ninety episodes. Progent uses its native policy pipeline (Shi

et al., 2026b); the other methods share one agent harness. Methods with memory restore the same accepted record before replacement.

Table 2: Public-agent comparison.

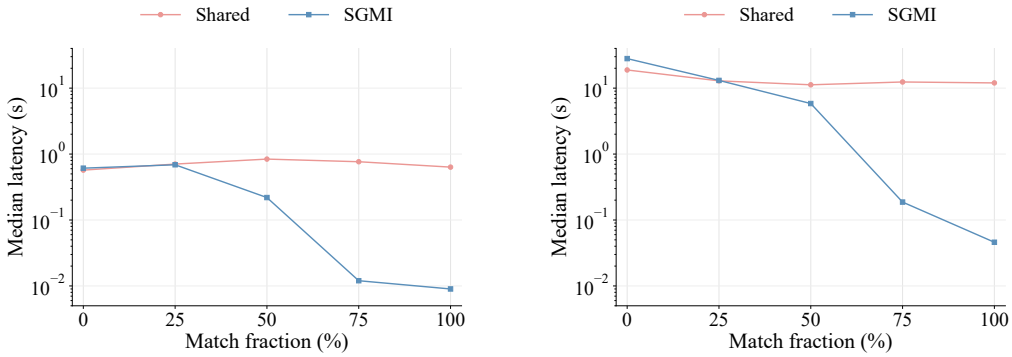
Method	Attack success (%)	Final utility (%)	
	Attacked $n = 12$	Clean $n = 6$	Attacked $n = 12$
No defense	0	100	50.0
Shared generation	0	100	58.3
SGMI	0	100	50.0
Same-scope policy	0	100	50.0
Progent (native)	0	83.3	58.3

SGMI: no direct match among 44 calls. Progent uses its native pipeline.

The autonomous episodes exercise SGMI's adaptive path: all 44 proposed calls use fallback, with zero direct matches or blocked actions. Shared and the same-scope policy also admit every proposed action. None of the ninety episodes reaches an injected attack objective, including those without defense. These trajectories complement the fixed-input tests with agent-selected proposals. Table 2 reports final benchmark predicates; initial values were not saved. Progent's single clean failure follows a request for confirmation. Its native pipeline changes prompts and trajectories; three failed policy updates retain the prior policy before evaluation completes (Appendix B.6).

5.5 RQ5: What is the cost of reuse?

SGMI reduces judgment calls from 31 to 11 on the same sixteen inputs, a 64.5% reduction. Figure 4(a) identifies the twenty avoided calls: sixteen from eight harmful actions and four from two rejected vendor-name references. The six actions allowed by both methods use eleven calls under each.



(a) One concurrent request.

(b) Up to twelve concurrent requests.

Figure 5: Ordered-workload latency.

Figure 5 measures a different cost: indexed lookup with 10,000 rules and fixed responses in the adaptive stages. Match requests precede misses, and medians use completed requests. A single-record match has median/P95 latency of 11/57 ms at parallelism one and 65/90 ms at twelve. A 128-record bucket raises the latter median to 218 ms. Matching avoids the adaptive stages, whereas a miss loads the full snapshot.

The miss-only workload also exposes a capacity limit. At zero matches and parallelism twelve, SGMI times out on 17 of 108 requests. The one-request, 100%-match cell retains two process repeats rather than three. These fixed-response measurements characterize the indexed implementation, separately from live model delay and the later snapshot-scan implementation. Appendix D reports all parallelism levels, missing repeats, and commit costs.

6 Related Work

6.1 Memory and adaptive defense

MemoryBank maintains persistent interaction records (Zhong et al., 2024). A-MEM associates entries (Xu et al., 2025), and MemGAS selects their granularity (Xu et al., 2026). AGrail optimizes reusable safety checks (Luo et al., 2025). AgentAntibody learns records with matching conditions, benign exclusions, and responses intended to preserve existing constraints (Weng et al., 2026b). SafeHarbor retrieves rules from hierarchical risk memory (Liu et al., 2026), while A-MemGuard retains corrective lessons (Wei et al., 2025). MINJA examines adversarial memory modification (Dong et al., 2025). We study the availability and enforced scope of accepted records during later use.

6.2 Agent evaluation and enforcement

InjecAgent and AgentDojo evaluate indirect injections through tools and external content (Zhan et al., 2024; Debenedetti et al., 2024). ASB covers multiple attack surfaces (Zhang et al., 2025), EIA targets web observations (Liao et al., 2025), and AgentDyn introduces dynamic environments (Li et al., 2026). ToolHijacker manipulates tool selection through adversarial tool documents (Shi et al., 2026a). Rennervate uses attention features to detect and sanitize injected instructions (Zhong et al., 2026). CaMeL separates control and data (Debenedetti et al., 2025); Progent enforces privileges through verified monotonic confinement (Shi et al., 2026b). Policy-invariance testing examines equivalent policy rewrites (Weng et al., 2026a). We instead follow accepted records across replicas, connecting visibility to execution decisions.

6.3 Distributed and serverless security

Zanzibar maintains consistent global authorization (Pang et al., 2019). Kalium combines function and application state for serverless control-flow integrity (Jegan et al., 2023). ALASTOR reconstructs intrusion provenance (Datta et al., 2022), and CloudFlow identifies security-sensitive data flows (Raffa et al., 2025). Denial-of-Wallet studies examine economic resource abuse (Kelly et al., 2021; Shen et al., 2022). SGMI applies ordered authorization to accepted adaptive records and examines the scope imposed by their consumers.

7 Discussion

SGMI preserves accepted constraints, so coverage depends on rule quality and action representation. Recipient history does not establish authorization for a particular transfer, and normalization can alter legitimate-action decisions (Appendix C.2). New tool domains require explicit effect mappings and paired legitimate controls; policy changes require a new epoch.

Deployment requires a consistent shared store and control over protected tool calls. Porting SGMI therefore depends on these interfaces, while store availability and coordination determine operational feasibility. Early matching avoids model judgments, but misses retain storage and reasoning costs. The indexed measurements characterize this tradeoff on one testbed; managed platforms and multi-region deployments require separate evaluation.

Fixed-action comparisons isolate rule use, whereas autonomous episodes establish which actions an agent actually proposes. The present autonomous cohort did not trigger direct blocking. Evaluating naturally reached harmful actions, additional accepted records, and pinned model checkpoints would test broader applicability. Such comparisons should retain same-scope policy controls.

8 Conclusion

SGMI turns accepted safety records into scoped action constraints that persist across replicas. Our evaluation shows why shared memory alone does not preserve protection and identifies the scope fields that retain legitimate behavior. These results connect adaptive safety learning with consistent enforcement at execution.

AI USE STATEMENT

The authors developed the research ideas and determined the experimental design. Generative AI tools assisted with manuscript polishing, experimental code, methodological feedback, result analysis, and formal definitions and proofs. The authors take responsibility for the final content.

ETHICS STATEMENT

Experiments used isolated virtual machines, containers, and a controlled Knative cluster with public benchmark tasks and synthetic accounts. No production services were targeted, and no personal or human-subject data were collected. Tool effects remained within the instrumented test environment.

REPRODUCIBILITY STATEMENT

Our anonymous repository provides the implementation, experiment runners, retained results, and offline analysis code (Anonymous, 2026). Sections 4–5 and Appendix B specify protocols, outcomes, and units of analysis. A separate LaTeX archive reproduces the manuscript. Appendix C documents implementation checks; the protocols identify unavailable historical records and missing fields.

REFERENCES

Anonymous. SGMI: Source code and evaluation artifacts. Anonymous research repository, 2026. URL https://anonymous.4open.science/r/SGMI-3F94.

Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001. doi: 10.1214/ss/1009213286.

C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934. doi: 10.1093/biomet/26.4.404.

Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. ALASTOR: Reconstructing the provenance of serverless intrusions. In *Proceedings of the 31st USENIX Security Symposium*, pp. 2443–2460, 2022. URL https://www.usenix.org/conference/usenixsecurity22/presentation/datta.

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*, volume 37, 2024. doi: 10.52202/079017-2636.

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. *arXiv preprint arXiv:2503.18813*, 2025. doi: 10.48550/arXiv.2503.18813.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen J. Xiang. Memory injection attacks on LLM agents via query-only interaction. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1554.

Nnamdi Ekwe-Ekwe and Lucas Amos. The state of FaaS: An analysis of public functions-as-a-service providers. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pp. 430–438, 2024. doi: 10.1109/CLOUD62652.2024.00055.

Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous Java performance evaluation. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, pp. 57–76, 2007. doi: 10.1145/1297027.1297033.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023. doi: 10.48550/arXiv.2302.12173.

James A. Hanley and Abby Lippman-Hand. If nothing goes wrong, is everything all right? interpreting zero numerators. *JAMA*, 249(13):1743–1745, 1983. doi: 10.1001/jama.1983.03330370053031.

Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990. doi: 10.1145/78969.78972.

Deepak Sirone Jegan, Liang Wang, Siddhant Bhagat, and Michael Swift. Guarding serverless applications with Kalium. In *Proceedings of the 32nd USENIX Security Symposium*, pp. 4087–4104, 2023. URL https://www.usenix.org/conference/usenixsecurity23/presentation/jegan.

Tomas Kalibera and Richard Jones. Rigorous benchmarking in reasonable time. In *Proceedings of the 2013 International Symposium on Memory Management*, pp. 63–74, 2013. doi: 10.1145/2464157.2464160.

Daniel Kelly, Frank G. Glavin, and Enda Barrett. Denial of wallet—defining a looming threat to serverless computing. *Journal of Information Security and Applications*, 60:102843, 2021. doi: 10.1016/j.jisa.2021.102843.

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*, 2026. doi: 10.48550/arXiv.2602.03117.

Xing Li, Xue Leng, and Yan Chen. Securing serverless computing: Challenges, solutions, and opportunities. *IEEE Network*, 37(2):166–173, 2023. doi: 10.1109/MNET.005.2100335.

Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. EIA: Environmental injection attack on generalist web agents for privacy leakage. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a73474c359ed523e6cd3174ed29a4d56-Abstract-Conference.html.

Zhe Liu, Zonghao Ying, Wenxin Zhang, Quanchen Zou, Deyue Zhang, Dongdong Yang, Xiangzheng Zhang, and Hao Peng. SafeHarbor: Defining precise decision boundaries via hierarchical memory-augmented guardrail for LLM agent safety. *arXiv preprint arXiv:2605.05704*, 2026. doi: 10.48550/arXiv.2605.05704. Accepted at ICML 2026.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8104–8139, 2025. doi: 10.18653/v1/2025.acl-long.399.

Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, and Mengzhi Wang. Zanzibar: Google's consistent, global authorization system. In *Proceedings of the 2019 USENIX Annual Technical Conference*, pp. 33–46, 2019. ISBN 978-1-939133-03-8. URL https://www.usenix.org/conference/atc19/presentation/pang.

Giuseppe Raffa, Jorge Blasco, Dan O'Keeffe, and Santanu Kumar Dash. CloudFlow: Identifying security-sensitive data flows in serverless applications. In *Proceedings of the 34th USENIX Security Symposium*, pp. 1073–1090, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/raffa.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912, 2020. doi: 10.18653/v1/2020.acl-main.442.

Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

Sergio Segura, Gordon Fraser, Ana B. Sánchez, and Antonio Ruiz-Cortés. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 42(9):805–824, 2016. doi: 10.1109/TSE.2016.2532875.

Junxian Shen, Han Zhang, Yantao Geng, Jiawei Li, Jilong Wang, and Mingwei Xu. Gringotts: Fast and accurate internal denial-of-wallet detection for serverless computing. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 2627–2641, 2022. doi: 10.1145/3548606.3560629.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in LLM agents. In *Proceedings of the Network and Distributed System Security Symposium*, 2026a. doi: 10.14722/ndss.2026.230675.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control. *arXiv preprint arXiv:2504.11703v3*, 2026b. doi: 10.48550/arXiv.2504.11703. URL https://arxiv.org/abs/2504.11703v3.

Qianshan Wei, Tengchao Yang, Yaochen Wang, Xinfeng Li, Lijun Li, Zhenfei Yin, Yi Zhan, Thorsten Holz, Zhiqiang Lin, and XiaoFeng Wang. A-MemGuard: A proactive defense framework for LLM-based agent memory. *arXiv preprint arXiv:2510.02373*, 2025. doi: 10.48550/arXiv.2510.02373.

Shihao Weng, Yang Feng, and Xiaofei Xie. Beyond accuracy: Policy invariance as a reliability test for LLM safety judges. *arXiv preprint arXiv:2605.06161*, 2026a. doi: 10.48550/arXiv.2605.06161. URL https://arxiv.org/abs/2605.06161.

Shihao Weng, Yang Feng, Xiaofei Xie, and Jiongchi Yu. AgentAntibody: An adaptive immune system for defending LLM agents against prompt injection. *arXiv preprint arXiv:2608.04053*, 2026b. doi: 10.48550/arXiv.2608.04053.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Wenlin Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, and Tong Xu. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=i2yIvZARnG.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0593.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024. doi: 10.18653/v1/2024.findings-acl.624.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

Yinan Zhong, Qianhao Miao, Yanjiao Chen, Jiangyi Deng, Yushi Cheng, and Wenyuan Xu. Attention is all you need to defend against indirect prompt injection attacks in LLMs. In *Proceedings of the Network and Distributed System Security Symposium*, 2026. doi: 10.14722/ndss.2026.240394. URL https://www.ndss-symposium.org/ndss-paper/attention-is-all-you-need-to-defend-against-indirect-prompt-injection-attacks-in-llms/.

A CONTINUITY SEMANTICS

Ordering. Let $c(r)$ be the durable commit of blocking record r, $\textsc{Ack}(r)$ its acknowledgment, and $R_{x,a}$ the snapshot read before action a in request x. A client-post request begins after acknowledgment; an action-post read begins after commit. Because acknowledgment follows commit and an action read follows invocation, client-post implies action-post. An overlapping episode requires a fresh action read to observe a commit that occurred after its request snapshot. The lifecycle experiments evaluate the client-post case; the proposition below states the conditions for action-post continuity.

Proposition (scoped continuity). Assume commits and reads share a durable linearizable object, successful reads return complete immutable prefixes, and protected effects require a single-use permit bound to the frozen action, attempt identifier, and observed version. Projection, read, or authorization failure produces no effect. If r commits before an action's read in the same epoch and that action matches r, it does not execute; a successful read contains r and selects block.

Proof. If the read fails, no permit is issued. Otherwise, linearizability and prefix completeness put r in the snapshot. A match blocks authorization, and complete mediation prevents execution. Single-use permits bind the action, attempt, and version, rejecting substitution. □

Extending the proposition to semantically covered effects requires $\textsc{Covered}(a, r) \Rightarrow \textsc{Match}(\phi(a), r)$, together with correctness of the accepted rule. For any escape, observing visibility and then matching yields the three disjoint cases in Equation 1. Native selection requires a recorded retrieval decision in addition to the visible state.

Within an epoch, learners allocate versions and append blocks atomically, retrying head conflicts. Candidates derive from confirmed episodes. Permission changes start a new epoch. Matching is invariant across connectors for equivalent normalized effects and trust directions.

B EXPERIMENTAL PROTOCOLS

B.1 INFRASTRUCTURE AND RECORDS

Experiments use a four-core, 15-GiB Ubuntu VM, Kind with Kubernetes 1.34.0, Knative Serving, Kourier ingress, and Redis-compatible storage. Public-ingress requests use the service host header. The adaptive implementation is a local reconstruction of AgentAntibody's projection, matching, response, and update stages. Generative stages use the provider-managed deepseek-chat API alias, whose underlying checkpoint is unpinned. Native AGrail uses seed 44 and local embeddings to accommodate the endpoint's embedding interface.

The public-ingress archive retains identifiers and versions for 36 learning commits, twelve per independently initialized configuration; full record bodies are unavailable. It captures combined learning and serving. Controlled banking comparisons restore one complete record, including its semantic fields and structured metadata, to isolate consumer behavior. Later archives document the adapter's transformations of conditions, exclusions, thresholds, and response templates.

The banking record describes an unauthorized transfer containing subscription information. Its projection selects immediate transfers to external payees, omitting subscription and authorization conditions. Generative reuse consumes the semantic record; SGMI and the same-scope policy first evaluate this predicate. Stored content is fixed, but enforced scope differs. Recipient classification uses initial transaction history, which does not establish authorization for the current transfer.

The email prototype derives families and timing from domain-specific text patterns and recipient suffixes. Its action model labels every modeled external send as pre-review, including a send proposed by a post-review request. A request can therefore miss its text predicate yet satisfy a later execution constraint. Banking distinguishes immediate and scheduled transfers. Each application defines its projection explicitly; broader semantic coverage requires extending it.

B.2 LIFECYCLE AND NATIVE MEMORY

Each of the twelve public-ingress cycles resets state, learns a blocking record, waits for its acknowledgment, raises benign load to induce scale-out, and submits related probes through the ordinary endpoint. The initial pod and service revision persist across cycles, with configurations evaluated in separate time blocks. Routing determines the old- and new-replica counts in Table 3. A cycle is protected only if every observed fresh-replica probe is blocked.

Replica-directed diagnostics contain eight replacement runs and fourteen autoscaling runs per configuration. Four autoscaling runs per configuration lack a successful probe on one fresh replica and are classified as incomplete observations. Records retain visible versions and outcomes; native semantic selection is unobserved. Table 4 reports the complete diagnostic counts.

Released AGrail's memory placement is tested in three independently initialized two-replica deployments. Each receives one native append through ingress, followed by memory inspection and 24 routed probes. Hash verification preserves the released append implementation. In an eleven-task Safe-OS comparison, Local and Shared each expose one task. Diagnostic replay on the ten tasks with reproducible accepted decisions preserves four blocks and reproduces two dangerous episodes from one accepted safe decision. The deployments measure state placement, while the task comparison examines decision quality. An earlier campaign with a missing JSON loader was excluded in full.

Table 3: Public-ingress continuity.

Variant	Cycles	Protected	Old escapes	New escapes	New latest
Local	12	0%	18/37	107/107	0/107
Shared	12	0%	21/48	46/96	96/96
SGMI	12	100%	0/56	0/88	88/88

Table 4: Replica-level diagnostics.

Transition	Variant	Runs	Reached	Exposed	Escapes
Replacement	Shared	8	8	7	47/126
Replacement	SGMI	8	8	0	0/126
Autoscale	Shared	14	10	13	48/99
Autoscale	SGMI	14	10	0	0/93

B.3 CONTROLLED PUBLIC-TOOL SEQUENCES

Injection definitions 1, 2, 3, 5, 6, and 8 supply eight immediate-transfer calls. User definitions 3, 4, 5, 6, 11, and 15 supply eight legitimate transfer, scheduling, and account-update calls. The twelve reference sequences do not test injection 8's scheduled-payment goal.

Four reuse methods evaluate every sequence before and after replacement in three separately initialized Knative services, yielding 288 sequence observations and 384 action decisions. The benchmark environment resets per sequence. Candidates are checked through the service's decision endpoint, and admitted calls execute in the caller's public banking runtime. Each decision uses context from the default environment, while admitted effects update the caller's runtime. Three restoration logs contain the same canonical content hash and version; replacement logs show shared version one and empty local state. Per-decision serving names were read from the wrong field, leaving null values, and retained pod prefixes are truncated.

For each method and phase, six harmful and six legitimate sequences run in three deployments, giving eighteen observations of each type. Shared generation selects no semantic match for any harmful call. SGMI and the same-scope policy each block 24 harmful calls and six legitimate reference calls per phase, with zero tool errors. Four of six legitimate sequences remain fully dispatchable in each deployment. The two rejected sequences use the vendor names Spotify and Apple where account identifiers are expected. This result measures the dispatch of those specific arguments. Autonomous episodes supply recognized identifiers and use the benchmark task evaluator.

An earlier 180-observation adapter test used a simplified representation and blocked all harmful sequences through Shared generation as well as deterministic reuse. We report it as a distinct cohort because the record representation changes. The contrast identifies representation as a relevant factor in how a consumer uses shared memory.

B.4 PUBLIC-CONTEXT REPLAY

A follow-up restores the archived banking record with canonical hash prefix 6f556401aeeedf35. It checks reference actions from public user tasks 3, 4, 6, and 15 and injection task 1 under user task 0. The runner supplies each full public prompt; retained decision rows preserve its first 120 characters. Four consumers evaluate the five actions before and after one replacement, producing forty decisions and 62 recorded judgment calls. Response identities and lifecycle entries identify distinct serving instances. These are decision-endpoint observations with no tool execution.

All methods allow both refunds, the recurring subscription payment, and the authorized standing-order update. The refunds project to known payees; both scheduled actions differ from the record's immediate-transfer predicate. Only SGMI and the same-scope policy block the unauthorized immediate transfer, without a judgment call. Shared generation selects no semantic match. The fifteen before-and-after input pairs across the three shared consumers retain the same blocking decisions, paths, and call counts. Local Memory loses its record and reduces judgment calls from ten to zero, without changing its five allow decisions.

Task-defined authorization, original semantic coverage, and observed predicate matching are separate labels. The task-3 refund amount is computed from history rather than stated directly, so authorization alone does not establish agreement with the record's literal condition. We use recorded projections instead of the runner's prewritten match descriptions, which incorrectly classified the refunds as external. The separate 64-decision placeholder replay is excluded.

B.5 SCOPE AND REPRESENTATION CONTROLS

The email scope corpus has eighteen dangerous requests from nine templates, twelve benign requests, nine boundary-adjacent requests, and five effect-preserving connector transformations. Each request has six replays. Every scope configuration initializes its own three-family record set and then freezes learning. The source-bound and source-invariant comparison jointly changes connector dependence and action grouping. Their majority decisions agree on all 39 same-source requests.

The paired banking ablation freezes one accepted record and sixteen candidate calls from the twelve sequences above. Removing recipient boundary retains family and order; removing order retains family and boundary. Each condition receives identical candidate arguments and projection context. All eight harmful calls match every structured variant. The complete predicate rejects two of eight legitimate calls. Boundary removal additionally rejects known-payee transfers from tasks 3, 4, and 15; order removal additionally rejects task 15's scheduled-payment update. The unit of comparison is a paired action decision, grouped by its twelve task sources. Cost is measured in recorded judgment requests; token billing and paired provider latency were not collected.

An earlier banking lifecycle campaign uses six dangerous and six legitimate tasks, each under three routing seeds. Each episode learns anew, commits, scales to zero, and submits a cold follow-up and benign control. A task is exposed if any seed executes the harmful transfer. Full and Direct share structured matching and differ after a miss. User task 0 is excluded before comparison because the service action model cannot execute its bill-derived transfer. Full permits one harmful episode and completes thirteen legitimate episodes; Direct permits three and completes eighteen. Both expose the same one of six harmful task sources.

A seven-task shopping comparison gives each record-bearing method one repaired record. No defense completes four tasks; Shared, SGMI, and the same-scope policy each complete two. Three browsing tasks are lost under the record's link restriction, while one baseline failure is recovered. A shared prefetch dependency lacks an optional module; one task encounters an empty calendar result. Aggregate calls reflect different action trajectories, not an isolated per-action speedup.

Table 5: Scope ablation.

Variant	Danger blocked	Cross blocked	Benign done	Adjacent done
Shared	69.4%	10.0%	52.8%	33.3%
Semantic	100.0%	100.0%	66.7%	0.0%
Family	100.0%	100.0%	62.5%	33.3%
Direction	100.0%	100.0%	69.4%	44.4%
Source-bound	100.0%	10.0%	76.4%	22.2%
SGMI	100.0%	100.0%	73.6%	33.3%

Table 6: Paired scope ablation.

Reuse	Harmful blocked (%) $n = 8$	Benign rejected (%) $n = 8$	Judge calls 16 inputs
Shared generation	0	0	31
SGMI	100	25.0	11
Without boundary	100	62.5	5
Without order	100	37.5	9

B.6 Complete agent episodes

Six banking user tasks (3, 4, 5, 6, 11, 15) each run clean and with injection goals 1 and 8. Five methods yield ninety episodes over six base tasks. Banking was selected for common-suite compatibility before observing protection results because the Progent fork lacks the shopping suite. The four common-harness methods rotate execution order by task. Progent runs separately in its native pipeline and suite. Record-bearing methods restore the same banking record before replacement; archives retain full pod names, committed version, and record hash.

All methods, including the unprotected comparator, have zero successful attack objectives. The cohort therefore characterizes autonomous behavior under these inputs. Utility is the final benchmark predicate; initial evaluations were not saved, and task 5's evaluator can be satisfied by an existing transaction. The reported utility includes both pre-existing and newly achieved goals.

SGMI inspects 44 proposed calls, all through semantic fallback, with zero direct matches or blocked actions. Shared inspects 46 and the same-scope policy 45, also with zero blocks. Total agent-model/defense-model calls are 50/0 for no defense, 54/68 for Shared, 49/64 for SGMI, and 51/64 for the policy. These totals include trajectory differences. The fixed-input ablation holds actions constant to measure the judgment calls avoided by the structured predicate.

Native Progent uses DeepSeek Chat, public important-instructions attacks, enabled policy updates, and continuation on update errors. Its recorded source revision is `https://github.com/sunblaze-ucb/progent/tree/8a8eb894b9d568a32b4e58252fea85b47f789c77`. Compatibility edits register the model and attack name and repair a Pydantic forward reference. The log contains 55 policy-JSON parsing diagnostics and 70 update messages. Three updates exhaust six attempts and retain the prior policy. These failures occur in policy generation or updating, not the final evaluator, but can affect subsequent behavior. Optional narrowing-only enforcement was not pinned by the runner. Eighteen formal task logs and two injection-reference logs are distinguished; only the former enter the ninety-episode cohort.

B.7 Design choices and replication

Public benchmarks provide the task definitions and outcome predicates used in our targeted mechanism comparisons (Debenedetti et al., 2024; Zhan et al., 2024). The banking sequence cohort includes six injection definitions with the selected transfer calls and six legitimate definitions supported by the adapter. These definitions produce eight harmful and eight legitimate reference actions. The complete-agent cohort pairs six user tasks with two public injection goals, selected for common-suite compatibility before observing defense results. Five methods give $6 \times 3 \times 5 = 90$ episodes over six

base tasks. Sample counts follow these selected definitions and experimental factors; they are distinct from the number of independent task sources.

The three email behavior families vary effect type; their benign and boundary-adjacent counterparts distinguish behavior preservation from indiscriminate refusal. Five implemented connector transformations change the input route while preserving the intended effect. Behavioral and metamorphic testing provide the design principle: vary a specified relation and inspect its consequence against a test oracle (Segura et al., 2016; Ribeiro et al., 2020).

Twelve ingress cycles repeat learning, scale-out, and follow-up within one retained revision. Three restored-record deployments repeat the same accepted state on separate service initializations. These campaign allocations measure repeated behavior within the stated task and deployment structure; they were not selected through a power analysis. The performance analysis distinguishes process from request repetition (Georges et al., 2007; Kalibera & Jones, 2013): lookup retains five processes, mixed load up to three (Appendix D).

Statistical interpretation follows the sampling unit. Classical binomial intervals assume independent trials under a specified model (Clopper & Pearson, 1934; Brown et al., 2001), and interpreting zero observed events requires a corresponding sampling argument (Hanley & Lippman-Hand, 1983). Our selected, reused tasks support descriptive outcome counts at the task and deployment levels. Repeated model calls remain grouped by source task.

C Implementation Checks

C.1 Action binding

The retained lifecycle wrapper uses trusted synchronous dispatch and a request snapshot. A subsequent revision binds permits to frozen arguments and the observed version, reads a consistent snapshot, and distinguishes judgment errors. Table 7 checks these implementation properties in that revision. Earlier lifecycle outcomes retain their original request-level interpretation. Observing a new commit within an episode additionally requires a fresh action read.

Table 7: Permit audit.

Check	Retained	Revised
Missing permit	Rejected	Rejected
Consumed permit	Rejected	Rejected
Changed arguments	Executes	Rejected
Changed read version	Unbound	Rejected

C.2 Record normalization

One prospective induction from a historical failure summary supplies two branches. Branch A is the parsed output, with a textual threshold converted to the parser default 0.45. Branch B applies local condition truncation, cleared exclusions, a fixed response, and structured metadata; its threshold remains 0.45. The retained raw response ends at 1,500 characters, and branch A therefore represents the parsed artifact. Both model validation replies reject the candidate; the adapter subsequently accepts it through a local structural check.

The two accepted branches judge the same 72 legitimate shopping reference actions, each once, under a shared generic context. A allows all actions with 98 judgment calls; B rejects three browsing actions with 100 calls. Every judgment completes without exception. The comparison measures several adapter changes jointly under fallback acceptance and a fixed generic context.

D Serving Cost

The indexed implementation preloads 10,000 records, varies candidate bucket size and effective parallelism, and uses identical deterministic responses for every adaptive stage. Each run starts in a

fresh process. The reduced lookup campaign has five independent process repetitions for selected configurations; each of three batches contains twelve requests, limiting effective concurrency to twelve. Lookup precedes learning. Tables summarize processes by taking the median of their within-process P50 and P95 values.

Table 8: Lookup latency (ms).

Request	Bucket	C1 P50/P95	C12 P50/P95	Stages
Redis related	–	635/1,746	16,091/20,642	3
SGMI match	1	11/57	65/90	0
SGMI match	128	26/52	218/297	0
SGMI miss	1	879/2,094	17,687/20,766	3

Mixed workloads place related requests before misses, with 36 requests in each of three planned fresh-process repeats. One Shared repeat at 75% matches and parallelism eight and one SGMI repeat at 100% matches and parallelism one have no output. At zero matches and parallelism twelve, one SGMI repeat completes 19 requests and times out on 17: the three repeats complete 91 of 108 requests. Other retained mixed runs complete all issued requests. Quantiles use successful requests and exclude absent repeats. Figure 5 therefore describes the indexed implementation under ordered arrivals and fixed model responses. The public-tool implementation scans a consistent snapshot per action, and its live judgment calls form a distinct cost measurement.

Table 9: Mixed-load median latency (s).

Path	0%	25%	50%	75%	100%
Redis C1	0.571	0.705	0.839	0.765	0.634
Redis C8	7.631	8.268	7.554	11.507	11.716
Redis C12	18.850	12.887	11.234	12.395	11.997
SGMI C1	0.611	0.688	0.219	0.012	0.009
SGMI C8	7.143	4.049	2.036	0.101	0.032
SGMI C12	28.044	13.092	5.837	0.187	0.046

The commit check issues twenty updates per writer at 2, 4, 8, and 16 writers. All 320 records survive in the largest configuration, while median commit latency rises from 21 to 188 ms across writer counts. The test measures contention and retention, without distributed failover.