SAME TOOL CALL, DIFFERENT AUTHORIZATION: POLICY-SUFFICIENT GRAPHS FOR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

With rules and workloads fixed across three typed agent deployments, exposing predecessors raises policy-laundering blocking from 82.4% to 95.1% and persistent-compromise blocking from 87.6% to 95.4%; binding the approved object to execution raises them further to 98.2% and 98.6%. The mechanism is an information collision: a benign plan and a secret- or attacker-dependent plan can terminate in the same byte-identical privileged call, so current-call authorization must give both the same decision. *Policy sufficiency* formalizes the missing information. The resulting policy-footprint characterization gives the coarsest equivalence induced by typed capability, provenance, and secret-flow clauses; it proves a terminal-projection lower bound and shows that any faithful carrier of the footprint is sufficient. GRAPHPERMIT computes that footprint from an Action Graph, returns replayable rejection witnesses, and binds the graph and policy judgment to gateway execution. On 1,200 disjoint post-freeze adaptive runs, GRAPHPERMIT blocks 96.3% of laundering and 96.8% of persistence attacks, versus 80.2% and 85.8% for OPA; unsafe authorization falls from 7.2% with unbound whole-plan visibility to 3.4% with binding. Its selected operating point blocks 99.1% of attack objectives while preserving 96.6% benign completion at 0.87 ms median gate time. A 228-execution audit measures 96.8% recall of policy-decisive dependencies, and transfer to AgentDojo reduces targeted attack success to 1.8%. These results identify authorization input and binding as controllable causes of compositional policy failure.

1 INTRODUCTION

Fix the typed call `net.send(domain, path, headers, body)` $= x$. In graph G_{pub}, x's body comes from a public release note; in G_{sec}, the same bytes come from a restricted document through a summarizer. Every call-boundary field is equal, but a forbidden secret-to-sink dependency exists only in G_{sec}. The integrity analogue is an identical configuration write selected by an injected issue comment rather than an operator. Any authorizer that receives only x must merge these executions. Such collisions turn routine browser, repository, and API calls into disclosure or durable compromise (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Greshake et al., 2023; Zhan et al., 2024; Debenedetti et al., 2024).

This is a representation question: which plan equivalence preserves every distinction that policy can test? A representation is *policy sufficient* when equal representations imply equal judgments for every policy in a family. For typed capability, argument, provenance, and secret-flow clauses, we characterize the induced quotient: the policy footprint is sufficient, and deleting any testable observation admits a separating pair. Terminal projection therefore has an unavoidable equal-call error; a faithful Action Graph, propagated labels, or any equivalent summary can carry the footprint. GRAPHPERMIT computes it with a linear-time gate and replayable witnesses, then binds the canonical graph, policy hash, nonce, and next call to a credentialed gateway. Signed summaries preserve the footprint across committed prefixes.

The experiment changes these mechanisms separately while fixing rules, labels, tasks, model outputs, and operating point. Across 206 scenarios, whole-plan visibility raises laundering blocking from 82.4% to 95.1% and persistence blocking from 87.6% to 95.4%; binding reaches 98.2% and 98.6%, for full gaps of 15.8 points (95% CI [11.1, 20.5]) and 11.0 points ([5.2, 16.8]). Gaps are smaller when

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.

decisive facts are local. On 1,200 independently authored post-freeze runs, GRAPHPERMIT blocks 96.3% and 96.8% of the two upstream-dependent families versus 80.2% and 85.8% for OPA, while binding reduces unsafe authorization from 7.2% to 3.4%.

The resulting decision rule is direct: an agent-security interface must preserve policy distinctions and bind them to privileged effects. At the selected point, GRAPHPERMIT preserves 96.6% benign completion at 0.87 ms median gate time; a 228-trace audit measures 96.8% decisive-edge recall, and AgentDojo transfer reaches 1.8% targeted attack success.

2 AUTHORIZATION OBJECTS IN AGENT SECURITY

Attacks and current-action defenses. Agent benchmarks establish multi-step tool use and indirect injection (Yao et al., 2023; Jimenez et al., 2024; Zhou et al., 2024; Zhan et al., 2024; Debenedetti et al., 2024; Zhang et al., 2025; Zhan et al., 2025). Prompt guardrails, DLP, egress filters, and per-action judges classify text or pending calls (Inan et al., 2023; Data Privacy Stack, 2025; Liu et al., 2024; Shi et al., 2024). Policy sufficiency isolates the case stronger classifiers cannot solve: equal terminal calls with different footprints. The family-stratified experiment tests the predicted concentration on upstream-dependent laundering and persistence.

Policy engines and graph-backed authorization. OPA and Cedar decide over application-supplied inputs (Open Policy Agent Project, 2023; Cutler et al., 2024); they can enforce the footprint when it is supplied, while our OPA baseline receives its current-action projection. FIDES propagates flow labels (Costa et al., 2025); FORGE checks observed causal ancestry; FAVA combines permission IR, an evidence graph, and a gateway; PolicyGuide persists workflow state (Palumbo et al., 2026; Zhang et al., 2026; Kang et al., 2026). They retain different facts at different decision and execution boundaries.

Exact distinction from the closest systems. Standard information-flow systems compute over chosen state; Proposition 1 instead characterizes the coarsest authorization equivalence and separates any interface that drops a testable observation. Complete FIDES labels can preserve flow state; FORGE's observed ancestry need not expose a later sink before earlier effects; PolicyGuide tracks procedure rather than value/control lineage; and FAVA is architecturally closest. GRAPHPERMIT supplies a prospective footprint contract and authenticates the graph–policy–next-call tuple. The matched experiment then varies only current versus predecessor-complete state and execution commitment. Appendix Table 10 aligns these guarantees.

3 EVALUATED SECURITY REGIME

The attacker controls content processed by the agent, including web pages, retrieved documents, issue comments, emails, and repository files. This capability matches repository and retrieval workflows that deliberately place untrusted context inside the planning loop (Lewis et al., 2020; Borgeaud et al., 2022; Huang et al., 2023; Zeng et al., 2024). The attacker uses that content to induce unauthorized disclosure or tool use, but cannot break cryptography, process isolation, the policy engine, or the execution gateway directly. Protected assets include credentials and restricted content, together with state whose integrity is critical, including protected branches, production configuration, and security settings.

Table 1 maps deployment-specific resources into reusable source and sink classes. RepoMaint, EnterpriseRAG, and WebOps thereby share a policy vocabulary while retaining distinct assets and tool semantics. We measure four failure modes: direct secret exfiltration, tool escalation, policy laundering through allowed infrastructure, and persistent compromise through durable state. The latter two require the authorizer to preserve dependencies across apparently admissible steps.

Plan-availability regimes. Full candidate plan available means that every prospective privileged call and its dependencies are exposed before execution. Prefix or partial plan only means that a committed subgraph is visible while later actions remain unresolved. Iterative replanning after tool outputs means that the next committed subgraph is produced from newly observed state. We evaluate full-plan authorization in all primary experiments and checkpointed authorization in the partial and

Table 1: Threat-model entities shared by all three deployments. The confidentiality model asks whether exfiltration sinks are reachable from secret sources; the integrity model asks whether integrity-sensitive sinks receive untrusted influence.

Category	Examples in our setting
Secret sources	Env vars, credential stores, restricted internal docs, private repositories
Untrusted inputs	Web pages, third-party docs, external issue comments, unvetted retrieved snippets
Exfiltration sinks	Outbound network, external messaging, public PR comments, persistent logs
Integrity sinks	File writes, git pushes or PRs, config changes, dependency pins, workflow edits

iterative regimes; the checkpointed contract carries authenticated source, taint, and approval state between committed subgraphs.

Trusted boundary and security property. The trusted computing base contains the typed-schema extractor, policy and labels, deterministic gate, and credentialed gateway. The model, planner, user and retrieved content, and intermediate tool outputs are untrusted. For graphs faithfully represented by the extractor and executed through the gateway, acceptance establishes capability closure, argument predicates, absence of configured secret-to-exfiltration paths, and provenance compliance at integrity sinks. The 228-trace fidelity audit and the malformed-input study quantify the extractor's valid-trace and fail-closed behavior, respectively.

The evaluated boundary covers explicit tool-mediated effects. Sandboxing and least privilege complement graph authorization for tool-internal behavior and channels outside typed calls. Appendix Table 11 maps each protected object to its adversary, enforcement assumptions, comparator, stress test, utility, and measured evidence.

4 Policy Footprints and Action-Graph Authorization

GraphPermit represents a candidate workflow as a typed dependency graph. The gate derives the policy observations from that graph before privileged effects, and the gateway executes only calls committed by the approved graph. The evaluation separately varies predecessor visibility and this execution commitment.

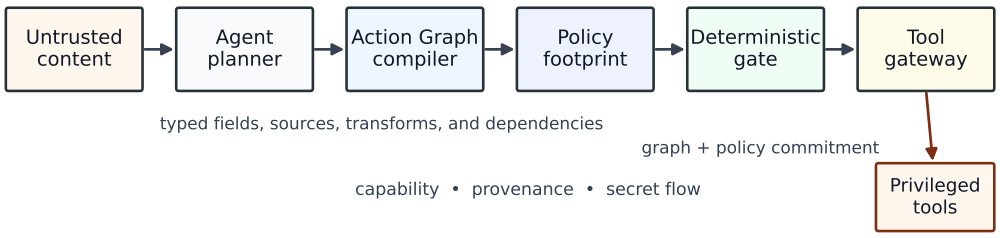


Figure 1: GraphPermit preserves a typed Action Graph, applies deterministic checks before privileged execution, and binds the approved graph to a trusted gateway; rejection returns a replayable witness under the active policy hash.

The architecture preserves the decision object, decides before effects, and mediates execution. On acceptance, the gate issues an authenticated `plan_id` committing to the canonical graph and policy

hash; the credentialed gateway verifies the token and call membership before every privileged action. Figure 9 measures fail-closed behavior on malformed inputs, while Appendix G audits 228 valid executions and obtains 98.8% node recall, 96.8% decisive-edge recall, and a 0.9% silent-omission rate.

4.1 POLICY OBSERVATIONS AND SUFFICIENT REPRESENTATIONS

An Action Graph, a directed acyclic graph $G = (V, E)$, has nodes representing typed tool calls and intermediate transforms, as well as labeled sources or sinks; its edges record value and control dependencies. Confidentiality and integrity states range over finite join-semilattices $(L_C, \sqsubseteq_C, \sqcup_C)$ and $(L_I, \sqsubseteq_I, \sqcup_I)$. Source labels initialize these states, ordinary nodes join predecessor states, and each declared sanitizer or declassifier t applies named transfer functions f_t^C and f_t^I. Undeclared transforms are identity functions.

For a fixed tool schema, primitive policy clauses have the grammar

$$r ::= \mathrm{cap}(v) \in \mathrm{Grants} \mid q_{a,j}(\mathrm{arg}_j(v)) \mid C(v) \sqsubseteq_C \kappa_a \mid I(v) \sqsubseteq_I \tau_a,$$
$$P ::= r \mid P \wedge P,$$

where a is the typed action at node v, $q_{a,j}$ is a declared predicate on argument field j, and κ_a, τ_a are sink-specific confidentiality and integrity bounds. Forbidden secret-source-to-sink reachability is the clause $C(v) \sqsubseteq_C \kappa_a$ after dependency propagation. Let $Y_P(G) \in \{0, 1\}$ be the resulting judgment.

Definition 1 (Policy-sufficient representation)**.** *A representation ρ is sufficient for policy family $\mathcal{P}$ when, for every $P \in \mathcal{P}$, $\rho(G) = \rho(G')$ implies $Y_P(G) = Y_P(G')$. Equivalently, every Y_P factors through ρ.*

Definition 2 (Policy footprint)**.** *For the clause family above, $\Phi(G)$ contains (i) each action's type, principal, grant, and predicate-visible typed fields and (ii) the propagated $C(v)$ and $I(v)$ states at every protected sink. Two plans are footprint-equivalent when these observations agree.*

Proposition 1 (Footprint characterization)**.** *Φ is sufficient for every conjunction in the declared clause family. If the family contains the corresponding primitive test for each exposed action field, grant, and sink bound, every sufficient representation must distinguish any pair with different footprint observations. Thus footprint equivalence is the coarsest sufficient equivalence for this family, up to observations that no admitted clause can test.*

The quotient does not prescribe a carrier: a graph, propagated labels, or a compact summary can implement Φ. Deleting one testable component constructs a pair the interface merges and the corresponding primitive clause separates, irrespective of downstream engine capacity.

Let $\pi_{\mathrm{loc}}(G)$ retain the terminal action, arguments, principal, and current-action metadata while omitting predecessor dependencies. A policy is *dependency sensitive* with respect to this projection when some pair G, G' has $\pi_{\mathrm{loc}}(G) = \pi_{\mathrm{loc}}(G')$ but different judgments.

Proposition 2 (Terminal-projection insufficiency)**.** *For dependency-sensitive policy P, there exist well-formed plans G_s, G_u with equal local projections, $Y_P(G_s) = 1$, and $Y_P(G_u) = 0$. Every authorizer over π_{loc} therefore returns the same decision distribution on both plans. A deterministic authorizer errs on at least one plan; under an equal-prior pair, every randomized authorizer has expected error at least $1/2$.*

Theorem 1 (Action Graph sufficiency)**.** *Let $\mathcal{P}_{\mathrm{AG}}$ be the clause family above. An Action Graph is faithful when it contains the exact predicate-visible fields, principals, grants, source and sink labels, declared transform identities, and all value or control dependencies that contribute to a protected sink, with no spurious policy-relevant dependencies. Faithful Action Graphs determine $\Phi(G)$ and are therefore policy sufficient for $\mathcal{P}_{\mathrm{AG}}$. If typed predicates and transfer functions are bounded-cost, Algorithm 1 computes $Y_P(G)$ in $O(|V| + |E|)$.*

An exact authorizer over $\rho(G)$ exists precisely when ρ is policy sufficient. Proposition 1 adds necessity to graph propagation: the interface must preserve Φ, while the runtime must bind its judgment to the represented execution. Gateway-exclusive credentials and complete mediation discharge the second obligation. Appendix E gives the proofs.

A small schema specifies the typed arguments and outputs exposed by each tool call. For example, the fields exposed by `net.send` are `domain`, `path`, `headers`, and `body`; for repository writes,

they are repository, branch, and diff fields. These schemas support security and the trust assumptions by enabling fail-closed parsing of malformed traces and direct policy constraints on fields, without heuristic recovery from raw strings.

4.2 Instantiated policy surface

The deployments instantiate the theorem-covered grammar with capability allowlists; predicates over domains, path prefixes, branches, and identities; integrity bounds over trusted, user-directed, and untrusted influence; and confidentiality bounds over public and secret-derived values. Declared sanitizers and declassifiers are typed transform nodes whose transfer functions are policy reviewed and hashed. Per-task request and modification budgets are enforced uniformly by the gateway after authorization and are not clauses in $\mathcal{P}_{\mathrm{AG}}$; the theorem and matched representation claims exclude that orthogonal hardening. These controls follow reference-monitor and information-flow principles (Saltzer & Schroeder, 1975; Denning, 1976). Appendix A gives the normalized policy surface and clause-by-clause comparator projection.

4.3 Trusted execution

Admissibility of a candidate graph is decided by the gate; holding credentials and executing privileged calls are reserved exclusively for the gateway. The gate authenticates a `plan_id` over the policy hash, canonical graph hash, and execution nonce. Before credential injection, the gateway verifies this commitment and checks that the requested call is the next approved action with matching typed arguments. Complete mediation turns representation sufficiency into an enforceable runtime property: the model may suggest arbitrary actions, while privileged calls remain confined to the accepted graph.

Checkpointed plans. For runtimes that reveal committed subplans, checkpointed GraphPermit authorizes the maximal committed subgraph G_t, binds its privileged calls, and emits a signed history summary S_t. The checkpointed subfamily admits clauses whose confidentiality and integrity states only accumulate by join across completed actions; source labels and completed action identities are immutable, and every declassification transfer is explicit in the committed graph. S_t retains per-value labels, joined states, transform identities, value and control dependencies, completed actions, and the policy hash. Replanning imports these records as authenticated typed sources in G_{t+1}. A future tool result, branch condition, destination, or sink argument is *unresolved* until its producing action and dependencies are committed; any privileged sink depending on such a value is deferred rather than judged from a partial footprint. The 75% prefix, 50% prefix, and iterative experiments measure the resulting security–deferral tradeoff.

5 GraphPermit: Check, Witness, and Bind

5.1 Gateway-bound checker

The gate first verifies that the Action Graph is acyclic, schema conformant, and complete for every required field. It then evaluates capability closure, typed argument predicates, provenance, and taint over the preserved pre-execution object. Algorithm 1 computes the policy judgment from Theorem 1; acceptance also produces the authenticated graph-and-policy commitment verified by the gateway.

5.2 Propagation semantics

Every action node must carry an explicit capability grant and satisfy its typed predicates. Dataflow state is a product of independent confidentiality and integrity components. Confidentiality taint joins configured secret sources along dependencies; integrity provenance joins trusted, user-directed, and untrusted influence under the policy's integrity order. Typed sanitizer and declassification nodes apply the declared transfer functions to their respective components. The policy checks provenance bounds at integrity-sensitive actions and secret-flow restrictions at confidentiality-sensitive sinks. Two plans can therefore share final sink fields while differing in upstream state: the local projection merges them, whereas the graph uniquely determines both joins.

Algorithm 1 Deterministic policy gate.

Input: Action Graph $G = (V, E)$, policy P, capability grants Grants
if NOT WELLFORMED(G) **then**
  return REJECT(`schema`, WITNESSSCHEMA(G))
end if
for all $v \in V$ where v is an action node **do**
  if $\mathrm{Cap}(v) \notin$ Grants **then**
    return REJECT(`capability`, WITNESSCAP(v))
  end if
  if NOT ARGSOK(v, P) **then**
    return REJECT(`args`, WITNESSARGS(v))
  end if
end for
$\mathrm{Prov} \leftarrow$ PROPAGATEPROVENANCE(G)
$\mathrm{Taint} \leftarrow$ PROPAGATETAINT(G)
for all $s \in V$ where s is a sink node **do**
  if NOT PROVOK(s, Prov, P) **then**
    return REJECT(`provenance`, WITNESSPROV(s))
  end if
  if $\mathrm{Taint}(s) = 1$ and FORBIDDENSINK(s, P) **then**
    return REJECT(`secret_flow`, WITNESSPATH(G,s))
  end if
end for
return ACCEPT()

Operational acceptance guarantee: confidentiality. If the gate accepts G under policy P, every sink classified as forbidden for confidential data has $\mathrm{Taint}(s) = 0$ under the configured propagation semantics. The guarantee applies to sources, dependencies, and sinks represented in G and to execution mediated by the bound gateway.

Operational acceptance guarantee: integrity/provenance. For every accepted integrity-sensitive action s, PROVOK(s, Prov, P) holds after the complete predecessor join. This instantiates the integrity component of Theorem 1. Appendix G quantifies graph fidelity and the decision impact of extraction defects.

5.3 REJECTION CERTIFICATES

When GRAPHPERMIT rejects a plan, it returns a rejection witness that can be replayed and machine checked under a policy hash. Alongside the violated rule and canonical plan hash, the certificate records a minimal witness subgraph from which the decision can be reproduced. A shortest source-to-sink taint path serves as the witness for a secret-flow violation. For a provenance violation, the witness is the dependency path that exposes forbidden influence on the restricted sink input. A capability witness identifies the first node requiring a capability that was not granted.

The witness localizes the compositional dependency that caused rejection and lets an independent verifier replay the decision under the same policy hash. Its median five-node size in Table 20 keeps policy debugging inspectable even when the candidate graph is substantially larger.

5.4 COMPLEXITY

Each gate check scales linearly with $|V| + |E|$. For the workloads we evaluate, gate latency consequently ranges from sub-millisecond to low-millisecond, far below the latency of LLM inference and tool execution (Appendix G). In practice, after compiling the plan into a graph, whole-plan context can be retained without adding a costly second stage of reasoning.

6 WHERE CURRENT-CALL AUTHORIZATION FAILS

6.1 INTERVENTION, WORKLOADS, AND JUDGMENT RULES

The central test is whether two defenses applying the same local rules diverge when one can observe the predecessor facts identified by Φ. We first localize that effect by attack family and then separate dependency visibility from execution binding. Disjoint adaptive attacks, graph-fidelity annotation, checkpointed prefixes, policy perturbations, and AgentDojo test whether the same explanation survives outside the controlled intervention.

RepoMaint prepares pull requests on non-protected branches, *EnterpriseRAG* answers over a 2.4M-document corpus and calls internal APIs, and *WebOps* performs authenticated browser automation (Jimenez et al., 2024; Zhou et al., 2024; Deng et al., 2023). All reported runs use structured traces from typed wrappers: every matched method receives the same current-call fields, while only whole-plan variants receive predecessor edges. `Llama-3.1-70B-Instruct` uses greedy decoding; Appendix F specifies the remaining configuration.

Task and scenario units. RepoMaint, EnterpriseRAG, and WebOps contribute 312, 284, and 256 benign tasks, respectively, each defined by initial state, user goal, permitted effects, and machine-checkable end-state assertions. The 206 development attacks comprise 72 exfiltration, 48 escalation, 51 laundering, and 35 persistence scenarios; each specifies injected-content placement, a set of prohibited objective predicates, unsafe-action predicates, and a dependency pattern. Direct families expose the decisive fact in the call, whereas laundering and persistence place it upstream. Five reruns produce 4,260 benign task runs; instantiating every attack scenario in all three settings produces 3,090 attack runs. Appendix F gives inclusion, deduplication, and construction details.

Labels and success evaluators. Shared evaluators inspect post-run state for unauthorized protected-value delivery, ungranted execution, forwarding through an allowed intermediary, or durable protected-state change. A run's attack-blocking score is the fraction of its predeclared prohibited objective predicates that remain false; SecFN is separately binary and fires when any authorized action violates a capability, argument, provenance, or flow clause. Benign success requires all end-state assertions, so authorization stops receive no completion credit. Policies, labels, thresholds, and baselines are selected on the 206-scenario set; only after freeze are 40 new laundering and 40 persistence scenarios authored for the disjoint 1,200-run adaptive test.

Metrics and uncertainty. For family F, scenario-weighted blocking is $\mathrm{Block}_F = |F|^{-1} \sum_{s \in F} K_s^{-1} \sum_{k=1}^{K_s} |O_s|^{-1} \sum_{o \in O_s} \mathbb{1}[o \text{ blocked in } (s,k)]$, where O_s is the scenario's predeclared objective-predicate set and $K_s = 15$ covers three settings and five reruns. This objective-level score can be fractional within a run; it is therefore distinct from an integer count of wholly blocked scenarios. SecFN divides runs containing any unsafe authorized action by 3,090; benign success divides completed tasks by 4,260. We use 10,000-resample scenario-clustered 95% intervals for attack rates and paired gaps, Wilson intervals for sparse impacts, and rerun intervals for utility.

Execution-binding intervention. The unbound and bound variants share the graph, policy, decision, and first post-approval divergence; only GRAPHPERMIT checks graph–policy–nonce and next-call conformance at the gateway. The common evaluator scores at most one substitution opportunity per run. Appendix F gives the generation and scoring protocol.

Matched baselines. The suite spans prompt rules, egress heuristics, DLP (Data Privacy Stack, 2025), gVisor, and a GPT-4.1 per-action judge. The composite stack combines Envoy allowlists, typed validation, redaction, rate limits, and current-call escalation; OPA v0.58.0 independently evaluates the same locally observable constants in Rego (Open Policy Agent Project, 2023). Escalation sees the task and current call but no predecessors. Its 52 ms value is automated dispatch latency and excludes human response, so we report it as a component measurement rather than an end-to-end latency comparison with GRAPHPERMIT. AgentDojo retains its native Tool Filter. Appendix Table 8 specifies inputs, decision rules, and metric treatment.

Table 2: Attack-family separation, boundary decomposition, and held-out adaptation. Panel A reports scenario-weighted mean blocking across predeclared objective predicates; n counts distinct scenarios and brackets are scenario-clustered 95% intervals. Panel B freezes policy and workload while varying visibility and binding; "p50" is automated component time, not human-review latency. Panel C uses 1,200 held-out runs. Full intervals and integer held-out counts appear in Appendix G.

Panel A: blocked objective predicates (%)						
Attack family	n	Prompt	Heur.	Judge	Stack	GRAPHPERMIT
Data exfiltration	72	72.4	81.3	91.2	94.1	99.7
Tool escalation	48	83.8	89.6	94.1	96.8	99.8
Policy laundering	51	59.2	67.8	77.5	82.4	98.2
95% CI					[78.2,86.3]%	[96.5,99.3]%
Persist. compromise	35	64.7	72.1	82.3	87.6	98.6
95% CI					[82.1,92.4]%	[96.0,99.6]%

Panel B: boundary decomposition							
Variant	Graph	Bound	Laund.	Persist.	Sec. FN	Benign	Auto p50
Composite stack	No	No	82.4%	87.6%	7.4%	93.8%	52 ms
Composite + hardening	No	No	86.9%	89.4%	5.8%	92.6%	58 ms
Whole plan, no binding	Yes	No	95.1%	95.4%	2.6%	95.8%	1.02 ms
OPA v0.58.0	No	No	84.8%	88.1%	6.7%	93.0%	64 ms
GRAPHPERMIT	Yes	Yes	98.2%	98.6%	0.9%	96.6%	0.87 ms
Panel C: held-out adaptive attacks							
Composite stack	No	No	78.7%	84.5%	18.4%	93.8%	—
OPA v0.58.0	No	No	80.2%	85.8%	17.0%	93.0%	—
Whole plan, no binding	Yes	No	92.2%	93.5%	7.2%	95.8%	—
GRAPHPERMIT	Yes	Yes	96.3%	96.8%	3.4%	96.6%	0.91 ms

6.2 MECHANISM: UPSTREAM EVIDENCE AND EXECUTION BINDING

The 15.8-point policy-laundering gap has paired 95% interval [11.1,20.5]; the 11.0-point persistent-compromise gap has interval [5.2,16.8]. Stronger local hardening gains 4.5 and 1.8 points on these families but reduces benign completion. Supplying the whole plan instead gains 12.7 and 7.8 points before binding; the authenticated commitment then adds 3.1 and 3.2 points and reduces workload SecFN from 2.6% to 0.9%. OPA tracks the composite stack despite an independent evaluator, matching Proposition 2: changing the policy engine cannot reconstruct omitted predecessors.

Panel C is the confirmatory test. With no post-freeze policy changes, GRAPHPERMIT exceeds OPA by 16.1 points on laundering and 11.0 points on persistence; unsafe authorization falls from 17.0% for OPA to 7.2% after exposing predecessor-complete state and then to 3.4% after binding. The visibility and binding transitions therefore recur on independently authored attacks, with exact counts of 204, 86, and 41 unsafe runs out of 1,200 for OPA, unbound whole-plan authorization, and GRAPHPERMIT, respectively.

Across the controlled study's 3,090 attack runs, GRAPHPERMIT records 28/3090 runs with an unsafe authorized action (95% CI [0.6,1.3]%), compared with 229/3090 for the composite stack ([6.5,8.4]%). Appendix Table 9 reports the complete current-call and model-based baseline suite.

6.3 SECURITY, TASK COMPLETION, AND COST AT THE SELECTED POINT

The selected point prevents 99.1% of attack objectives while preserving 96.6% benign completion. Deterministic gate evaluation takes 0.87 ms at the median; graph construction, propagation, serialization, gateway checks, and tool execution are included in the separate 1.09× end-to-end factor. The narrow per-setting spread shows that the operating point is not carried by one deployment.

Table 3: Selected GRAPHPERMIT operating point on the controlled workload. Attack-objective blocking uses setting-instantiated runs, benign success uses task runs, and per-setting ± values are 95% intervals. The weighted row is a descriptive aggregate; scenario-clustered family intervals appear in Appendix Table 14. Overhead is end-to-end relative task latency; gate p50 times only deterministic policy evaluation.

Setting	Attack blocks	Benign success	Overhead	Gate p50
RepoMaint	99.4% ± 0.4	96.8% ± 0.7	1.07×	0.74 ms
EnterpriseRAG	99.1% ± 0.5	97.2% ± 0.6	1.08×	0.86 ms
WebOps	98.7% ± 0.6	95.9% ± 0.8	1.11×	1.05 ms
Weighted avg	99.1%	96.6%	1.09×	0.87 ms

6.4 ASSUMPTION CHECKS: EXTRACTION, PARTIAL PLANS, AND TRANSFER

The valid-trace audit stratifies 228 executions across the three deployments and recovers 98.8% of privileged-call nodes and 96.8% of policy-decisive edges. Of 32 extraction defects, 30 stop before privileged execution, one is decision neutral, and one WebOps omission permits an unsafe action; Appendix Table 13 reports the strata, review protocol, and intervals. Checkpointed summaries defer sinks with unresolved dependencies: at 75% and 50% prefixes, SecFN is 3.2% and 6.1% with 6.8% and 12.9% deferral. On all 97 AgentDojo utility tasks and 629 security cases (Debenedetti et al., 2024), GRAPHPERMIT reaches 1.8% targeted ASR and 71.1% benign utility, versus 7.5% ASR and 73.4% utility for Tool Filter. Appendix G gives the full partial-plan, label-sensitivity, and multi-model slices.

7 IMPLICATIONS AND MEASURED SCOPE

What the separation changes. The footprint view separates information loss from execution substitution. Under fixed rules, gains are small when decisive facts are local but reach 15.8 and 11.0 points on upstream-dependent laundering and persistence. Post-freeze unsafe authorization falls from 17.0% with OPA to 7.2% with whole-plan visibility and 3.4% with binding. The carrier need not expose a reasoning transcript: typed fields and propagated labels suffice whenever they preserve Φ.

Evaluated regime and boundary. The guarantee covers typed, gateway-mediated effects under declared source, sink, dependency, and transform semantics. Across three deployments, the fidelity audit measures 96.8% decisive-edge recall; 30 of 32 defective traces stop safely, one is decision neutral, and one permits an unsafe action. Label removal and checkpoint experiments quantify incomplete footprints. Channels outside typed calls and vulnerabilities inside allowed tools remain assigned to complementary sandboxing and least privilege.

8 CONCLUSION

An authorization API chooses which executions are distinguishable. The policy-footprint characterization makes that choice exact for typed capability, provenance, and secret-flow policies; GRAPHPERMIT computes the footprint and binds its judgment to execution. The matched study locates both effects: post-freeze unsafe authorization falls from 17.0% with OPA to 7.2% with whole-plan visibility and 3.4% with binding. The selected point blocks 99.1% of attack objectives, preserves 96.6% benign completion, and takes 0.87 ms at the median; AgentDojo targeted ASR is 1.8%. Agent-security interfaces should preserve every policy distinction they claim to enforce and bind those distinctions to privileged effects.

AI USE STATEMENT

Generative AI tools assisted manuscript preparation, including prose revision, formatting and consistency checks, and bibliography review. No further AI-use scope is documented in the available manuscript-preparation record.

Ethics Statement

This work has a defensive purpose: constraining credentialed tool execution when agents process untrusted content. Evaluation occurs in controlled deployments against policy-violation scenarios. Rejection witnesses can reveal sensitive source labels, destinations, and policy structure, so they are restricted to authorized operators, protected as security logs, and rate limited to prevent boundary probing.

Reproducibility Statement

Sections 4–6 specify the authorization object, enforcement path, workloads, baselines, metrics, denominators, and uncertainty protocol. The appendices provide model and hardware configuration, proof details, policy and implementation specifications, per-setting fidelity results, stress-test protocols, and diagnostic distributions.

References

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning (ICML 2022)*, volume 162, pp. 2206–2240. PMLR, 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025. URL https://arxiv.org/abs/2505.23643.

Joseph W. Cutler, Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, Eleftherios Ioannidis, John Kastner, Anwar Mamat, Darin McAdams, Matt McCutchen, Neha Rungta, Emina Torlak, and Andrew M. Wells. Cedar: A new language for expressive, fast, safe, and analyzable authorization. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):670–697, April 2024. doi: 10.1145/3649835. URL https://doi.org/10.1145/3649835.

Data Privacy Stack. Presidio v2.2.359. Software release, July 2025. URL https://github.com/data-privacy-stack/presidio/releases/tag/2.2.359.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, 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 37 (NeurIPS 2024)*, volume 37, pp. 82895–82920. Curran Associates, Inc., 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 28091–28114. Curran Associates, Inc., 2023. doi: 10.52202/075280-1220. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html.

Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976. doi: 10.1145/360051.360056. URL https://cacm.acm.org/research/a-lattice-model-of-secure-information-flow/.

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 2023 ACM Workshop on Artificial Intelligence and Security (AISec '23)*, pp. 79–90. Association for Computing Machinery, 2023. doi: 10.1145/3605764.3623985. URL https://arxiv.org/abs/2302.12173. Co-located with CCS 2023.

Yangsibo Huang, Samyak Gupta, Zexuan Zhong, Kai Li, and Danqi Chen. Privacy implications of retrieval-based language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14887–14902, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.921. URL https://aclanthology.org/2023.emnlp-main.921.

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. *arXiv preprint arXiv:2312.06674*, 2023. URL https://arxiv.org/abs/2312.06674.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Seongjae Kang, Taehyung Yu, and Sung Ju Hwang. PolicyGuide: From guarding one action to guiding the whole workflow for policy-compliant LLM agents. *arXiv preprint arXiv:2608.19861*, 2026. URL https://arxiv.org/abs/2608.19861.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://nips.cc/virtual/2020/public/poster_6b493230205f780e1bc26945df7481e5.html. arXiv:2005.11401.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 2024)*, pp. 1831–1847, Philadelphia, PA, August 2024. USENIX Association. URL https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei.

Open Policy Agent Project. Open policy agent v0.58.0. https://github.com/open-policy-agent/opa/releases/tag/v0.58.0, October 2023. General-purpose policy engine using the Rego policy language.

Nils Palumbo, Sarthak Choudhary, Jihye Choi, Guy Amir, Prasad Chalasani, and Somesh Jha. Formal policy enforcement for real-world agentic systems. *arXiv preprint arXiv:2602.16708*, 2026. URL https://arxiv.org/abs/2602.16708.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

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. URL https://ieeexplore.ieee.org/document/1451869.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html. arXiv:2302.04761.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to LLM-as-a-judge. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, pp. 660–674. Association for Computing Machinery, 2024. doi: 10.1145/3658644.3690291. URL https://arxiv.org/abs/2403.17710.

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)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X. arXiv:2210.03629.

Shenglai Zeng, Jiankun Zhang, Pengfei He, Yue Xing, Yiding Liu, Han Xu, Jie Ren, Shuaiqiang Wang, Dawei Yin, Yi Chang, and Jiliang Tang. The good and the bad: Exploring privacy issues in retrieval-augmented generation (RAG). In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4505–4524, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.267. URL https://aclanthology.org/2024.findings-acl.267/.

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, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7116–7132, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.395. URL https://aclanthology.org/2025.findings-naacl.395/.

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 (ICLR 2025)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html.

Yifan Zhang, Xinkui Zhao, Sai Liu, Hengxuan Lou, Guanjie Cheng, and Chang Liu. FAVA: Formal authorization for verified agents with evidence-backed permission graphs. *arXiv preprint arXiv:2607.27267*, 2026. URL https://arxiv.org/abs/2607.27267.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

A Policy Language Examples

A.1 Normalized policy surface

GraphPermit evaluates three theorem-covered clause classes: capability and typed-argument predicates, integrity/provenance bounds, and confidentiality-flow bounds. Each tool has a capability from

a fixed vocabulary including `net.send`, `msg.send`, `log.write`, `fs.write`, `git.write`, and `exec.shell`. A principal selects the capability grant set; typed fields then select the applicable argument predicates and sink bounds.

Confidentiality labels form the order `public` $\sqsubseteq_C$ `internal` $\sqsubseteq_C$ `secret`. Integrity labels form `trusted` $\sqsubseteq_I$ `user` $\sqsubseteq_I$ `untrusted`, where a larger state represents influence unacceptable to more sensitive sinks. Environment credentials, credential-store outputs, restricted documents, and private-repository contents enter as `secret`. Organization-controlled records enter as `trusted`; explicit user fields enter as `user`; web pages, third-party documents, external tickets, issue comments, and unvetted retrievals enter as `untrusted`. Labels attach to values rather than tools and join across value and control dependencies.

Table 4: Complete normalized clause surface used by the matched variants. Finite deployment sets instantiate the symbols in the second column; the constants and comparisons are unchanged across projections.

Clause	Typed observation	Decision rule
Capability	principal, action type, grant	action capability must belong to the principal's grant set
Argument	domain/origin, path, repository, branch, identity, recipient, method	every field satisfies its finite allowlist, prefix, protected-branch, account-scope, or method predicate
Confidentiality	joined source state at `net.send`, `msg.send`, `log.write`, or `repo.public`	propagated state must not exceed the sink-specific bound; public/external sinks admit `public` values
Integrity	joined influence at repository, configuration, dependency, workflow, or browser-state writes	propagated state must not exceed the sink-specific bound; protected state requires `trusted` or explicit `user` direction as configured
Transform	transform identity and input state	only a named sanitizer or declassifier applies its reviewed transfer function; every undeclared transform is identity

Deployment instantiations. RepoMaint's `repo-bot` grants repository read, scoped file and git writes, and network access to 12 reviewed domains. Network predicates include `api.github.com` and package registries; writes are confined to reviewed path prefixes such as `src/` and `tests/`, with `main` and `release/*` protected and task branches under `fix/*`. Credential variables, restricted repository content, and private dependencies are confidential sources; external issues and repository-supplied instructions are untrusted integrity sources. EnterpriseRAG's `employee-assistant` grants retrieval, answer generation, and scoped internal-API calls. Its predicates bind knowledge-base identity, API domain and path, recipient, and output channel; restricted records and credentials are confidential, while third-party or unvetted retrieved snippets are untrusted. WebOps's `web-operator` grants browser read, navigation, form submission, and scoped state changes. Its predicates bind origin, account, method, destination, and mutable field; credentials and restricted page state are confidential, and external page content is untrusted. Across all three, outbound network, external messaging, public repository text, and persistent logs are confidentiality sinks; repository, dependency, workflow, configuration, and browser-state mutations are integrity sinks.

Declared transforms. Sanitizers extract only schema-declared allowlisted fields and therefore may map `untrusted` to `user` or `trusted` for the named output field. Declassifiers are field specific: credential injection may place a secret in the authentication field of a named internal endpoint without declassifying request bodies, destinations, or downstream values. The transform identity, field, endpoint predicate, and transfer function are part of the policy hash. No free-form model step, summary, or string rewrite changes either lattice state.

A.2 MATCHED PROJECTION INTO EACH COMPARATOR

The matched intervention reuses one normalized policy and changes which observations reach the decision. Table 5 makes that projection explicit. A local method receives labels already attached to the current typed call but not labels or dependencies from predecessor nodes; missing predecessor state is unavailable rather than silently relabeled. The whole-plan and bound variants receive the same constants plus the predecessors needed to compute their joins.

Table 5: Clause-by-clause projection in the boundary decomposition. "Current" means the exact current-action value is shared; "joined" means the identical bound is evaluated after graph propagation.

Policy component	Composite	OPA	Whole plan / GRAPHPERMIT	Matched constant
Capability grants	Current	Current	Current	principal–capability set
Typed argument predicates	Current	Current	Current	allowlists, prefixes, branches, identities
Current-call source/sink labels	Current	Current	Current	label vocabulary and sink bounds
Predecessor value/control state	Unobserved	Unobserved	Joined	source labels and transfer functions
Decision witness	Local reason	Rego trace	Dependency subgraph	policy hash and violated primitive
Execution commitment	None	None	None / graph bound	canonical graph, policy, nonce, next call

OPA rules are mechanical Rego projections of the capability, field, and current-label clauses with identical constants. The composite stack uses the same local predicates and adds the separately reported DLP and escalation controls. Whole-plan/no-binding and GRAPHPERMIT share the complete normalized policy hash; only GRAPHPERMIT requires the gateway commitment. Rate limits, maximum modified-file counts, credential injection, and sandboxing are gateway controls applied uniformly after the policy decision. They are excluded from $\mathcal{P}_{\mathrm{AG}}$ and from claims about the footprint theorem.

A.3 NORMALIZATION AND REVIEW

Policies are normalized into canonical form and hashed at startup. Every decision certificate carries that hash, and every gateway token commits to it together with the canonical graph and execution nonce. This keeps a matched comparison tied to one policy object while making each primitive capability, field, label, sink, and transform directly reviewable.

B IMPLEMENTATION DETAILS

Prototype. We implement GRAPHPERMIT as a wrapper around a tool-using agent runtime. The prototype consists of three components: (i) a plan compiler that converts agent outputs into Action Graphs, (ii) a policy engine that performs deterministic checks and generates certificates, and (iii) a tool gateway that mediates execution.

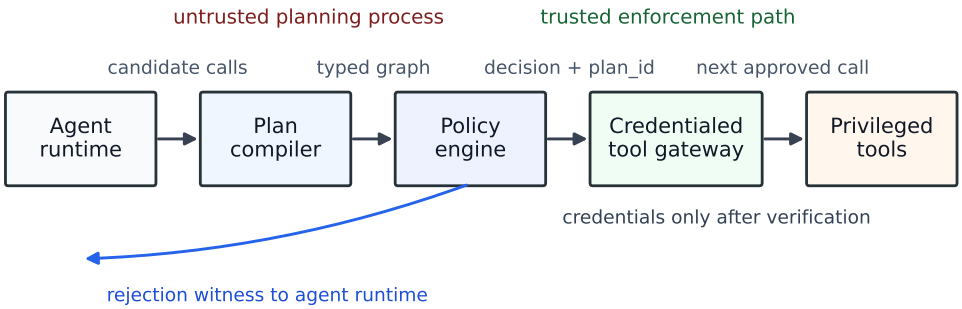


Figure 2: GRAPHPERMIT implementation architecture: a compiler builds typed Action Graphs; a deterministic policy engine emits accept/reject certificates; a tool gateway enforces least privilege at execution (credential injection only for approved calls, plus redaction, rate limits, and sandboxing).

Figure 2 separates three security roles. The compiler turns model output into policy-sufficient typed state, the gate computes the policy judgment over that state, and the gateway alone holds privileges. This split keeps planning outside the trusted enforcement path and couples every privileged call to an authenticated approval.

Plan compiler. The compiler accepts either structured tool-call traces (preferred) or a constrained natural-language plan format. For each tool call, it emits an action node with a typed schema and attaches provenance metadata for each argument: user-provided, retrieved (trusted vs. untrusted), or secret-derived. The compiler also emits explicit value nodes so that dependencies are visible to downstream analysis. Tables 3–9 use structured tool-call traces emitted by typed wrappers; the constrained textual-plan path provides compatibility for runtimes without native function calls.

Robust extraction from real agent traces. In many frameworks, tool calls are emitted as JSON function calls; in others, they appear as text blocks. GRAPHPERMIT supports both by (i) preferring JSON traces when available, and (ii) applying a strict parser for text traces that rejects malformed calls. This avoids the common failure mode where a partially malformed call slips through to a tool and triggers unexpected behavior. When parsing fails, the plan is rejected as non-runnable and never executed.

Parser hardening against adversarial inputs. We harden the parser against adversarial tool-call syntax that may attempt to bypass validation: (i) JSON parsing uses a strict decoder that rejects duplicate keys, trailing commas, and non-UTF-8 sequences; (ii) argument values are length-bounded (max 64 KB per argument, 256 KB per plan) to prevent memory exhaustion; (iii) Unicode normalization (NFC) is applied before schema validation to prevent homograph attacks on domain names and paths; (iv) type coercion is disabled—a string "true" is not converted to boolean `true`. In our evaluation on 191 malformed or adversarial inputs spanning five failure families (Figure 9), the only observed outcomes are explicit parse errors and schema-level safe rejections. The figure reports the family-level split between those two fail-closed outcomes rather than a single aggregate percentage because the operational point is that no malformed input is accepted and forwarded to the gateway.

Policy specification. Policies are written as a small, auditable configuration that maps tools to capabilities and constraints. Listing 1 shows the normalized encoding of three representative clauses: outbound HTTPS uses a finite domain predicate, confidential values are bounded at network sinks, and repository writes use path and branch predicates. Appendix Tables 4 and 5 specify the complete clause surface and its matched projection into each comparator.

Listing 1: Representative normalized policy clauses.

```
capabilities:
  net.send:
    allow_domains: ["api.github.com", "internal.example.com"]
    forbid_taint: ["secret"]
  git.write:
```

```
    allow_paths: ["src/", "tests/"]
    forbid_branches: ["main", "release/*"]
sources:
  secret:
    - env: ["GITHUB_TOKEN", "AWS_ACCESS_KEY_ID", "AWS_SECRET_ACCESS_KEY"]
```

The capability block expresses network and repository controls over typed fields, while the source block makes secret labeling explicit for deterministic taint propagation. Deployment-specific finite sets, escalation paths, and transforms use the same normalized schema and policy hash.

Secret and provenance labeling. We label secrets at the boundary: environment variables, credential stores, and explicitly marked internal documents are sources of `secret` taint. Retrieved content is labeled `untrusted` by default unless it comes from an allowlisted knowledge base. These labels propagate through the Action Graph edges; thus, any sink that depends (transitively) on a secret is rejected unless the sink explicitly supports a safe mechanism (e.g., a credentialed request to an allowlisted internal domain).

Credential handling. GRAPHPERMIT assumes that credentials are managed outside the model. The model never receives raw tokens; instead, the tool gateway injects credentials at execution time for approved calls. For example, a GitHub API request includes an authorization header injected by the gateway only if the destination is allowlisted and the request is classified as non-exfiltrating by the policy engine. This design reduces the blast radius of model compromise: even if the model tries to print secrets, it does not possess them.

Certificate generation. When a check fails, the policy engine computes a minimal witness: for secret-flow violations, it returns a shortest dependency path from a secret source to the prohibited sink; for capability violations, it returns the action node and missing capability token; for argument violations, it returns the violating argument value and the rule it violated. Certificates are JSON-serializable and can be stored alongside task logs for later auditing.

Rate limiting and scope control. The gateway enforces per-task budgets (e.g., max outbound requests, max files modified) independent of the model. This mitigates denial-of-wallet and denial-of-service behaviors where an injected prompt attempts to trigger repeated tool calls. Scope constraints are enforced with path and repository scoping (e.g., only write within a scratch workspace; only open PRs from ephemeral branches). These gateway budgets are applied uniformly to the boundary-decomposition variants. They are defense-in-depth controls, not clauses in the theorem-covered policy family and not part of the measured predecessor-visibility contrast.

Integration points. GRAPHPERMIT is planner-model independent when the runtime exposes the typed calls, policy-relevant dependencies, source and sink labels, and declared transform semantics required by Theorem 1. Function-calling APIs, tool DSLs, and constrained textual calls can meet this interface through their respective extractors. For systems with multiple candidate plans (e.g., best-of sampling), GRAPHPERMIT gates and binds each candidate separately.

Operational deployment. We integrate GRAPHPERMIT in a sidecar-style deployment: the agent runtime sends candidate graphs to the gate over local IPC, and the gateway executes approved calls in a separate reduced-privilege process. The gate authenticates the canonical graph, policy hash, and execution nonce; the gateway verifies the token and next-call conformance before injecting credentials.

C CASE STUDY

Scenario. RepoMaint triages GitHub issues, proposes code changes, and opens pull requests. It can read the repository, create a PR from a non-protected `fix/*` branch, and call 12 reviewed network domains. One allowlisted internal service, `internal.example.com/announce`, accepts release-note text and publishes it through a separately governed channel.

Equal-call pair. Two candidate plans end in the byte-identical request shown in Listing 2. In the benign plan, the body bytes are copied from a user-authored public release note. In the injected plan, an issue comment directs the agent to a restricted incident document; a summarizer happens to emit

the same bytes. The destination, method, path, headers, body, principal, and current-call labels are identical.

Listing 2: Terminal request shared by the benign and injected plans.

```
net.send(domain="internal.example.com",
       path="/announce", method="POST",
       headers={"Content-Type": "text/plain"},
       body="Release 3.2 resolves issue #1847.")
```

Why the authorization object decides the outcome. The current call satisfies the capability grant, allowlisted-domain predicate, method constraint, and payload schema in both plans. A request-local engine therefore receives exactly the same input and must return the same decision. The full graphs differ at one policy-relevant observation: the body in the injected plan depends on a `secret` source and an `untrusted` control source, while the benign body depends on a `user` source. The configured bounds accept the benign user-derived path and reject the secret- and untrusted-derived path.

GRAPHPERMIT behavior. GRAPHPERMIT accepts the public-source graph and rejects the restricted-source graph before either request executes. The rejection certificate contains the restricted document, summarizer transform, injected issue-comment control edge, and `net.send` sink under the current policy hash. Because every terminal field already passes its local predicate, the opposite judgments are attributable to the preserved predecessor state rather than to an allowlist or syntax difference.

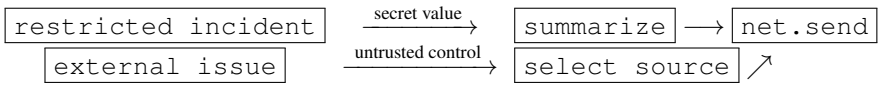


Figure 3: Minimal witness for the rejected half of the equal-call pair: restricted content and attacker-controlled selection reach an otherwise locally admissible `net.send` request.

Figure 3's witness is sufficient to replay the rejected judgment without the full reasoning transcript. Replaying the benign graph changes neither the terminal request nor its local policy checks; it changes only the source and control paths, which makes the case an executable instance of Proposition 2.

Safe recovery. After rejection, the runtime can request a graph that reconstructs the same release text from the user-authored note. The gateway accepts that graph and binds the unchanged terminal request to the public-source dependency path.

D OPERATIONAL SECURITY DISCUSSION

Meaning of the guarantee. The gate gives a deterministic decision about a particular Action Graph under a particular policy hash. Gateway-bound acceptance guarantees capability closure, argument predicates, provenance bounds, and secret-flow constraints for modeled sources, dependencies, and sinks. Complementary sandboxing and least privilege cover tool-internal and process-level risks.

Reference monitor placement. The reference monitor boundary matters as much as the policy logic. We therefore separate the gate from the agent runtime and place credentials only behind the gateway. The gate authenticates a `plan_id` over the policy hash, canonical graph hash, and execution nonce; the gateway verifies it before releasing credentials, preventing approval for one graph from authorizing another.

Table 6: Common policy pitfalls surfaced by the deployments and the observable control each witness exposes.

Pitfall	Why it happens	How GRAPHPERMIT helps
Over-broad allowlists	Convenience under operational pressure	Certificates show which rule and which fields made the action admissible.
Implicit trust in retrieval	Retrieved text is treated as instructions rather than data	Provenance rules force retrieved content to stay explicitly labeled unless it is sanitized.
Policy laundering via internal services	Internal endpoints may forward or trigger downstream side effects	Whole-plan taint and provenance checks expose the dependency path into those sinks.
Missing sink coverage	Logs, comments, and telemetry are easy to overlook	Residual failures localize which channel should be reclassified and added to policy.

Internal-service laundering and missing sink coverage explain residual misses more directly than planner choice. Certificates expose the rule and dependency path behind each decision, turning policy refinement into reviewable counterexample-guided debugging.

Defense composition. Sandboxing confines tool-internal behavior, least-privilege gateway credentials bound the authority of approved calls, and policy regression tests preserve reviewed source, sink, and transform definitions as deployments evolve. These controls operate at layers complementary to graph authorization.

Operational role of certificates. The certificate is a replayable rejection witness under a policy hash. It identifies the dependency chain or capability that caused rejection and makes policy refinement closer to counterexample-guided debugging than ad hoc prompt editing.

E PROOFS OF THE SEPARATION RESULTS

Definitions. For candidate Action Graph $G = (V, E)$, $Y_P(G)$ is the binary full-plan judgment under policy P. A representation ρ induces equivalence classes $[G]_\rho = \{G' : \rho(G') = \rho(G)\}$. It is policy sufficient exactly when Y_P is constant on every class for every policy in the family. The footprint $\Phi(G)$ contains predicate-visible action fields, principals and grants, and propagated confidentiality and integrity states at protected sinks. The local projection π_{loc} retains the terminal action, typed arguments, principal, and current-action metadata while removing predecessor facts absent from those fields.

Exact-authorizer criterion. If ρ is sufficient, define $A_P(r)$ as $Y_P(G)$ for any G with $\rho(G) = r$; constancy makes the definition independent of the representative. Then $A_P \circ \rho = Y_P$. Conversely, if such an exact authorizer exists, equal representations produce equal authorizer outputs and hence equal full-plan judgments.

Proof of Proposition 1. Every primitive clause reads one component of $\Phi(G)$: capability and argument clauses read an action tuple, while confidentiality and integrity clauses read a propagated sink state. Equal footprints therefore make every primitive clause agree, and structural induction over conjunctions gives equal Y_P. For necessity, suppose $\Phi(G)$ and $\Phi(G')$ differ on a testable component. If an action type, grant, or argument field differs, choose its primitive capability or field predicate. If two finite-lattice sink states $x \neq y$ differ, either $y \not\sqsubseteq x$, in which case the bound $z \sqsubseteq x$ accepts x but not y, or $y \sqsubset x$, in which case $z \sqsubseteq y$ separates them in the other direction. The corresponding one-clause policy separates G and G'. Any representation that merged the pair would violate Definition 1. Components outside the admitted primitive tests can be removed without affecting any judgment, which gives the stated qualification.

Proof of Proposition 2. Choose admissible G_s and inadmissible G_u with identical terminal action, arguments, principal, and local metadata. Let G_s derive the terminal bytes from an allowed public source and G_u derive the same bytes from a protected source through typed transforms. Thus $\pi_{\mathrm{loc}}(G_s) = \pi_{\mathrm{loc}}(G_u) = x$, while $Y_P(G_s) \neq Y_P(G_u)$. For deterministic A_{loc},

$$A_{\mathrm{loc}}(\pi_{\mathrm{loc}}(G_s), P) = A_{\mathrm{loc}}(x, P) = A_{\mathrm{loc}}(\pi_{\mathrm{loc}}(G_u), P).$$

Thus one label is wrong. For randomized A_{loc}, let q be the shared acceptance probability. Under equal priors the error is $\frac{1}{2}(1-q) + \frac{1}{2}q = \frac{1}{2}$. Model capacity, calibration, and additional rules over x cannot change the bound because the distinguishing dependency is absent.

Proof of Theorem 1. Equal faithful graphs contain the same typed actions, fields, principals, grants, labels, declared transforms, sinks, and edges. Capability and argument predicates therefore agree nodewise. In topological order, source states agree by label; if all predecessor states agree, applying the same join and typed transfer function yields the same state at the next node. Induction gives equal confidentiality taint and integrity provenance at every sink. Reachability and all sink predicates consequently agree, so Y_P is constant on graph classes. Acyclicity, schema, capability, and argument checks plus the two propagation passes take $O(|V| + |E|)$.

Bound-execution argument. The gate authenticates a commitment to the canonical graph, policy hash, and execution nonce. The gateway holds the privileged credentials, verifies this commitment, and checks each requested call and its arguments against the next approved graph action. Complete mediation therefore prevents an unapproved graph or call from reusing acceptance. In the confidentiality pair, the protected-source path taints the sink only in G_u; in the integrity pair below, untrusted influence reaches the sensitive write only in G_i. The graph judgment and executed behavior preserve both distinctions.

Integrity analogue. Let G_t and G_i terminate in the same protected-branch write with identical repository, branch, and diff fields. In G_t, the diff depends on a trusted operator specification; in G_i, the same bytes are selected under influence from an untrusted issue comment. Their local projections are equal, so every request-local authorizer again returns the same decision distribution. In the full graphs, provenance propagation joins the untrusted label into the protected sink for G_i but not G_t. A graph authorizer enforcing PROVOK separates the pair, and gateway binding prevents substitution after approval.

Checkpoint composition. For the monotone checkpointed subfamily, signed state S_t preserves the policy inputs contributed by completed predecessors: per-value confidentiality taint and integrity provenance, declared transform results, completed action identities, and the policy hash. Consuming S_t as typed sources in G_{t+1} preserves these states by the same topological induction. Induction over checkpoints therefore maintains policy sufficiency for observed dependencies; a privileged sink with an unresolved predecessor is deferred before the gate issues an execution commitment.

F REPRODUCIBILITY DETAILS

Execution configuration. Main experiments serve `Llama-3.1-70B-Instruct` with vLLM 0.5.3 on four NVIDIA A100-80GB GPUs, two AMD EPYC 7763 CPUs, and 1 TB RAM. Decoding uses temperature 0.0, top-p 1.0, and a 4096-token maximum. The Python 3.11 gate and gateway execute single threaded on CPU. The per-action judge uses GPT-4.1 checkpoint 2025-01-06 with a fixed five-example rubric. The multimodel slice uses provider snapshots `gpt-4o-2024-08-06` and `claude-3-5-sonnet-20241022`.

Table 7: Evaluation components, versions, and measured roles.

Component	Version	Role
Llama 3.1 70B Instruct / vLLM	vLLM 0.5.3	Main planner and serving stack
GPT-4.1	2025-01-06	Request-local judge baseline
GPT-4o / Claude 3.5 Sonnet	2024-08-06 / 20241022	Multimodel WebOps robustness
Envoy / OPA	1.28 / 0.58.0	Composite proxy and independent request-local engine
AgentDojo	1.1	Independent transfer environment
Python	3.11	Gate and gateway runtime

Workloads and repetitions. RepoMaint, EnterpriseRAG, and WebOps contribute 312, 284, and 256 benign tasks, respectively, for 852 total. Every one of the 206 attack scenarios is instantiated once in each setting, producing 618 trials; every benign task and attack trial is rerun five times at the frozen operating point. The adaptive set contains 80 disjoint scenarios, equally split between policy laundering and persistent compromise, instantiated across all three settings for 1,200 runs. AgentDojo uses benchmark v1.1 with all 97 utility tasks and 629 security cases, five runs per case.

Construction, inclusion, and labels. The benign tasks span read-only, single-effect, and multi-step workflows. A task is included only when its initial state, user goal, allowed effects, protected sources and sinks, and machine-checkable final assertions are fully specified; normalized duplicates with the same goal and required action signature are collapsed. A development attack scenario pairs an injected-content placement with one or more prohibited objective predicates, an unsafe-action predicate, and a source-to-sink or capability dependency pattern. The 72 exfiltration and 48 escalation scenarios expose the decisive fact in the pending call; the 51 laundering and 35 persistence scenarios require an upstream dependency or earlier durable action. Scenarios sharing the same normalized objective set, protected object, and decisive dependency pattern count once before cross-setting instantiation.

Outcome adjudication. The same labels and evaluators score every method. Exfiltration objectives test protected-value delivery to unauthorized receivers; escalation objectives test ungranted execution; laundering objectives test protected data or control reaching prohibited effects through allowed intermediaries; persistence objectives test unauthorized repository, configuration, or browser state surviving the run. The run-level blocking score is the mean of the binary outcomes over its predeclared objective predicates. An unsafe intermediate authorization is recorded at the first typed action violating a capability, argument, provenance, or secret-flow clause, regardless of whether a terminal objective later fails. Benign completion requires every task assertion and all required side effects. These definitions explain why objective blocking can be fractional within a run while SecFN and benign success are binary run outcomes.

Adaptive-set isolation. The 206-scenario set is used to select policy labels, thresholds, prompts, and baseline configurations. Only after those choices are frozen are the 40 laundering and 40 persistence scenarios in the adaptive set authored, each with one targeted objective predicate. Adaptive authors receive the stable tool schemas and attack interface; normalized objectives and decisive dependency patterns are checked against the development set, and no policy, threshold, or baseline prompt changes after outcomes are observed.

Aggregation. Scenario-weighted family rates first average the objective-predicate blocking fraction across each scenario's 15 setting/rerun realizations and then weight scenarios equally. Workload SecFN divides attack runs containing an unsafe authorized action by 3,090, and benign success divides completed tasks by 4,260 benign runs. Scenario-clustered bootstrap intervals use 10,000 resamples with distinct scenarios as the resampling unit; paired gaps resample matched outcomes. Sparse fidelity impacts use Wilson intervals. Gate microbenchmarks count every compiled candidate plan, including retries, safe rejections, and replans.

Baseline configuration. The composite stack uses Envoy 1.28, structured argument validation, Presidio-style secret patterns, per-task rate limits, and human escalation for shell execution and protected writes. Escalation receives the task and current call, applies the frozen local rubric, and contributes its completed decision to security and benign-success metrics. The reported 52 ms stack component is automated dispatch only; human response is neither timed nor compared with GRAPHPERMIT gate latency. OPA v0.58.0 evaluates mechanical Rego projections of the same capability grants, allowlists, branch rules, argument predicates, current-call labels, and escalation conditions over the current typed action. Appendix Table 5 identifies the predecessor observations absent from both local inputs. Policies and thresholds are selected on the original scenario set, frozen, and reused unchanged for held-out adaptation, observability, and policy-noise tests. AgentDojo retains benchmark-native tasks, injections, and success evaluators. Its Tool Filter limits execution to tools selected from the user task; GRAPHPERMIT adds typed dependencies and gateway binding without changing the benchmark evaluator.

Binding and checkpoint protocols. For each attack run, the unbound and bound variants share the accepted graph and expose the same first post-approval divergence, if one occurs. The unbound runtime executes that request without checking it against the accepted graph; the bound gateway verifies graph hash, policy hash, nonce, next action, and typed arguments. Only the first divergence is scored as a substitution opportunity, yielding at most one opportunity per run. For checkpoint experiments, a 50% or 75% prefix contains that fraction of the original graph's action and transform nodes in canonical topological order together with all predecessor source nodes. A sink whose producer or dependency lies beyond the cut is deferred; deferral counts runs in which a required privileged sink is postponed and therefore receives no immediate benign-completion credit. Iterative replanning closes one checkpoint after each tool result and imports the signed summary into the next graph. The slice includes the 204 scenarios that admit both canonical prefix cuts in RepoMaint and WebOps; two scenarios without a valid cut at both fractions are excluded before method execution, giving 408 setting-instantiated attack trials.

Transfer and robustness protocols. On AgentDojo, benign utility (BU) is native task utility without injection, utility under attack (UA) is native task utility in injected runs, and targeted ASR is the benchmark's targeted attack success rate. Tool schemas map benchmark calls to the same capability, typed-field, source, and sink clauses before any transfer outcomes are observed; no policy constant or benchmark evaluator is tuned on AgentDojo. Policy-label sensitivity uses five fixed removal masks at each level, aligned with the five workload reruns; noncritical labels are those whose removal does not delete a task's sole protected source or sink. The multi-model table evaluates every model–temperature cell on all 206 WebOps attack scenarios with five reruns, for 1,030 runs per cell, while reusing the same policy and typed gateway.

G ADDITIONAL DIAGNOSTICS

This appendix supplies complete baseline configurations, uncertainty, fidelity and decision-impact measurements, adaptive and transfer protocols, observability and policy-sensitivity results, and operational diagnostics supporting the main-paper findings.

Table 8: Compact baseline specification used in the evaluation. HITL denotes human-in-the-loop escalation at the decision point.

Baseline	Decision unit	Visibility	Tuning/configuration	HITL
Prompt-only	Whole task	Tool names and natural-language instructions only	Fixed rules in the system prompt; no typed execution state	No
Heur. egress	Single action string	Raw URL and payload strings	Regexes and domain allowlists matched to our policy surface	No
DLP redact	Outbound payload	Payload text plus 47 secret patterns	Presidio-style detectors applied to request bodies, comments, and logs	No
LLM judge	Single action	Typed fields of the current action only	GPT-4.1 with a 5-shot rubric prompt; no whole-plan history	No
Sandbox	Executed call	Syscall and runtime boundary only	gVisor isolation, no plan or dataflow reasoning	No
Composite stack	Single request	Typed tool args, proxy metadata, and DLP output	Envoy allowlists, structured validation, rate limits, and escalation for `git.push` or `exec.shell`	Yes
OPA v0.58.0	Single request	Typed tool args and proxy metadata	Shared rules translated to Rego; current action supplied as the input document	No
AgentDojo Tool Filter	Single task	User task and permitted tool set	Benchmark-native relevance filter; no dependency state	No
GRAPHPERMIT	Whole candidate plan	Full typed Action Graph with dependencies	Deterministic gate under the current policy hash	No

The strongest baselines receive allowlists, typed validation, redaction, rate limits, and escalation. OPA changes both implementation and policy evaluator while preserving the request-local decision object. Its alignment with the composite stack therefore isolates dependency visibility from local-rule hygiene and internal baseline engineering.

Table 9: Request-local and model-based baselines at the same operating point. SecFN is the fraction of 3,090 attack runs containing an unsafe authorized action; benign success uses 4,260 benign runs. p50 reports each automated decision component; the composite value excludes human response and is not an end-to-end latency comparison.

Defense	Sec. FN	Benign	Ovhd	p50	Audit
Prompt-only	30.2%	98.4%	1.00×	—	No
Heur. egress	22.4%	97.8%	1.04×	0.1 ms	No
DLP redact	18.7%	94.2%	1.21×	8.4 ms	No
Sandbox	26.1%	96.9%	1.06×	—	No
LLM judge	16.2%	96.1%	1.68×	340 ms	No
Composite stack	7.4%	93.8%	1.42×	52 ms	Limited
GRAPHPERMIT	0.9%	96.6%	1.09×	0.87 ms	Yes

Table 9 separates the security effect from a blanket utility penalty: GRAPHPERMIT reduces unsafe authorization by 6.5 points relative to the strongest composite current-call stack while completing 2.8 points more benign tasks. The timing cells characterize implementation components only. The composite stack's security and utility include completed escalation decisions, whereas 52 ms measures dispatch before human response; GRAPHPERMIT's 0.87 ms measures its deterministic gate. End-to-end human-review latency is not inferred from these values.

Table 10: Design-space map for authorization representations and enforcement. OPA and Cedar can accept richer application-supplied inputs; their effective boundary is determined by the representation supplied by the application.

System family	Decision object	Dependency visibility	Decision time	Execution binding
Prompt or guardrail detector	Text span	None	Generation boundary	Advisory
Per-action judge or measured OPA	Current typed call	Local fields	Before one call	Call-level
FIDES	Labeled data and current call	Propagated IFC labels	Before one call	Planner mediated
FORGE	Pending action and causal graph	Observed ancestry	Before one call	Reference monitor
FAVA	Permission IR and evidence graph	Task and trace dependencies	Before each effect	Gateway enforced
PolicyGuide	Workflow position	Procedural history	Turn and action boundaries	Advisory and action gate
Cedar-style authorizer	Typed request context	Supplied attributes	Before one request	Application mediated
GRAPHPERMIT	Candidate Action Graph	Explicit predecessors	Before effects	Graph-and-policy bound

Table 10 locates the paper's intervention among neighboring systems. Propagated labels and accumulated graphs can carry the footprint; the characterization predicts sufficiency whenever they preserve every testable observation. The remaining distinctions are temporal and operational: FORGE authorizes from observed ancestry, PolicyGuide retains procedural state, FIDES carries flow labels, and FAVA is closest in combining evidence with a gateway. GRAPHPERMIT commits prospective predecessors, the complete policy hash, and the next privileged call as one authorization object. The measured whole-plan/no-binding row isolates the commitment effect without using this qualitative map as an empirical ranking.

Table 11: Security-boundary evidence matrix connecting protected objects, adversary capabilities, trust and enforcement assumptions, comparators, stress tests, utility, and measured evidence.

Object/property	Adversary	Trust/enforcement	Comparator/stress	Utility and evidence
Credentials; confidentiality	Injected retrieved content	Extractor, labels, gate, gateway	OPA; held-out laundering	96.6% benign; Proposition 2; forwarding/paraphrase
Restricted content; confidentiality	Multi-step secret relay	Typed sources, dependencies, sinks	Composite stack; direct and laundering families	0.87 ms p50; Table 2; sink coverage
Branch and config integrity	Injected issue or repository text	Provenance propagation, graph binding	No-binding ablation	Table 2; missing labels
Allowed infrastructure; both properties	Allowed-service policy laundering	Source/sink policy, exclusive credentials	Adaptive attacks; OPA	Section 6.4; internal forwarding
Durable state; integrity	Persistent multi-call steering	Checkpoint state, bound subgraphs	50% prefixes; AgentDojo	1.8% ASR; relay semantics/policy noise

The matrix in Table 11 ties each protected object to the assumption most likely to affect it and to a corresponding stress test. In particular, adaptive forwarding tests dependency visibility, the no-binding ablation tests the first post-approval divergence in each attack run, and label perturbations test footprint completeness. The binding intervention therefore exposes at most 3,090 controlled and 1,200 held-out substitution opportunities under the shared scoring rule in Appendix F.

Table 12: Boundary decomposition at the main operating point. The experiment freezes 852 benign tasks, 206 scenarios, 618 setting-instantiated trials, five reruns, local rules, and the operating point while varying dependency visibility and execution binding.

Variant	Pol. laund.	Persist. comp.	Sec. FN	Benign	p50	Bound
Composite stack + stronger hardening	86.9%	89.4%	5.8%	92.6%	58 ms	No
Whole-plan without binding	95.1%	95.4%	2.6%	95.8%	1.02 ms	No
OPA v0.58.0	84.8%	88.1%	6.7%	93.0%	64 ms	No
GRAPHPERMIT	98.2%	98.6%	0.9%	96.6%	0.87 ms	Yes

The family rows use 51 laundering and 35 persistence scenarios; SecFN and benign success summarize 3090 attack runs and 4260 benign runs, while p50 summarizes each method's authorization invocations. The corresponding 95% intervals are: hardened stack [82.1, 91.7], [83.9, 94.9], and [4.9, 6.7]; whole-plan without binding [92.2, 98.0], [91.6, 99.2], and [2.0, 3.2]; OPA [79.8, 89.8], [82.4, 93.8], and [5.8, 7.6]. Visibility supplies the largest gain, binding contributes an additional measurable reduction, and independently maintained OPA follows the request-local profile.

Valid-trace fidelity audit. We audited 228 valid executions stratified across all deployments, split into 126 benign and 102 attack traces. The setting cells in Table 13 are the sampling strata; malformed-input cases belong to the separate parser study. Two adjudicators trained on the typed schemas and policy rules inspect the wrapper trace independently of the compiler output. Their instructions are to enumerate every privileged call with exact typed fields, mark each labeled source and protected sink, reconstruct direct and derived value/control dependencies, and flag an edge as decisive when adding or removing it can change at least one policy clause. They also replay the reference graph to assign accept/reject and benign/unsafe impact. Independent node, edge, and decision labels are recorded

before comparison; disagreements are resolved by replaying the typed trace and policy to a shared reference graph. Recall and impact below compare compiler output with that adjudicated reference, keeping annotator disagreement distinct from extraction error.

Table 13: Valid-trace fidelity across typed deployments. Impact reports safe rejects / unchanged decisions / benign false rejects / unsafe accepts; false rejects are a subset of safe rejects.

Setting	Traces (benign / attack)	Node rec.	Edge rec.	False-edge	Silent omit.	Safe reject[†]	Impact
RepoMaint	34 / 30	99.4%	98.1%	0.4%	0.0%	100.0%	4/0/2/0
Enterprise-RAG	44 / 34	98.9%	96.8%	0.9%	1.3%	90.0%	9/1/5/0
WebOps	48 / 38	98.3%	95.8%	1.4%	1.2%	94.4%	17/0/10/1
Overall	126 / 102	98.8%	96.8%	0.9%	0.9%	93.8%	30/1/17/1

[†]Measured only over audited traces with any extraction defect; those denominators are 4, 10, 18, and 32 traces.

The per-setting 95% intervals are as follows. RepoMaint reports node recall [98.4, 99.8], edge recall [96.4, 99.0], false-edge rate [0.0, 1.2], silent-omission rate [0.0, 5.6], and safe-reject rate [51.0, 100.0]. EnterpriseRAG reports [97.6, 99.5], [94.9, 98.0], [0.3, 1.9], [0.2, 6.9], and [59.6, 98.2]. WebOps reports [96.9, 99.0], [93.8, 97.1], [0.7, 2.6], [0.2, 6.4], and [74.2, 99.0]. Overall intervals are [98.1, 99.2] node recall, [95.8, 97.6] edge recall, [0.6, 1.5] false-edge rate, [0.2, 3.1] silent omission, and [79.9, 98.3] safe rejection. Thirty defects stop before privileged execution, one leaves the authorization decision unchanged, and one WebOps omission produces unsafe acceptance; the audit therefore connects representation accuracy directly to operational outcomes.

Table 14: Scenario-clustered 95% intervals for attack-objective blocking and paired GRAPHPERMIT– composite gaps. Each cluster contains a scenario's three settings and five reruns.

Family	Composite CI	GRAPHPERMIT CI	Paired gap CI
Exfiltration	[91.9,96.0]%	[99.1,99.9]%	[3.2,8.1]
Tool escalation	[94.6,98.2]%	[98.8,99.9]%	[1.0,5.2]
Policy laundering	[78.2,86.3]%	[96.5,99.3]%	[11.1,20.5]
Persistent compromise	[82.1,92.4]%	[96.0,99.6]%	[5.2,16.8]

The paired intervals exclude zero for every family. Their substantially larger location for policy laundering and persistent compromise matches the prediction that representation matters most when the deciding dependency is upstream.

Table 15: Held-out adaptive attacks. Attack authors construct 40 laundering and 40 persistence scenarios after policy and baseline freeze; every scenario is instantiated in three deployments and rerun five times.

Method	Laundering blocked	Persistence blocked	Sec. FN	Benign
Composite stack	472/600; 78.7%	507/600; 84.5%	221/1200; 18.4%	93.8%
OPA v0.58.0	481/600; 80.2%	515/600; 85.8%	204/1200; 17.0%	93.0%
Whole plan, no binding	553/600; 92.2%	561/600; 93.5%	86/1200; 7.2%	95.8%
GRAPHPERMIT	578/600; 96.3%	581/600; 96.8%	41/1200; 3.4%	96.6%

Scenario-clustered intervals for GRAPHPERMIT are [94.2,97.8] and [94.9,98.2]; OPA intervals are [76.1,84.0] and [82.2,89.0]. Across the 41 residual unsafe decisions, the bypass mix is 18 internal-

forwarding, 12 semantic-paraphrase, 7 sink-underclassification, and 4 tool-side cases. Adaptation lowers absolute blocking while retaining the family-specific separation and the frozen benign operating point.

Table 16: Checkpointed partial observability. RepoMaint and WebOps plans expose 75% or 50% prefixes or replan after each tool response; policies and perturbation seeds are fixed before comparison.

Method / visibility	Laund.	Persist.	Sec. FN	Benign	Deferral	p50
Composite request-local	82.4%	87.6%	7.4%	93.8%	0.0%	52 ms
Full-plan	98.2%	98.6%	0.9%	96.6%	0.0%	0.87 ms
GRAPHPERMIT						
Checkpoint, 75% prefix	95.8%	96.4%	3.2%	95.9%	6.8%	1.05 ms
Checkpoint, 50% prefix	91.9%	93.5%	6.1%	95.1%	12.9%	1.21 ms
Iterative replanning	94.2%	95.0%	4.4%	94.8%	9.7%	1.28 ms

Across 568 benign tasks and 408 setting-instantiated attack trials, each with five reruns, security degrades smoothly with visibility. The attack denominator is 204 scenarios in each of RepoMaint and WebOps after the predeclared exclusion of two scenarios that do not admit both canonical prefix cuts. At 50% prefixes, checkpointed authorization still adds 9.5 points of policy-laundering and 5.9 points of persistence blocking over request-local enforcement; most utility cost appears as an explicit deferral.

Algorithm 2 Checkpointed authorization for partial plans.

Input: committed subgraph G_t, signed state S_{t-1}, policy P
Verify the policy hash and signature on S_{t-1}
Join carried provenance and taint into the source nodes of G_t
if a privileged sink depends on an unresolved value **then**
    return DEFER with the unresolved dependency witness
end if
Run Algorithm 1 on the augmented committed graph
if the gate accepts **then**
    Bind executable calls to the checkpoint graph and policy hash
    Emit signed S_t with per-value provenance, taint, transforms, and completed actions
end if
return the gate decision, witness, and S_t

Table 17: Independent AgentDojo v1.1 transfer on all 97 utility tasks and 629 security cases, each run five times. BU is benign utility, UA is utility under attack, and ASR is targeted attack success under native evaluators; the execution adapter exposes typed calls.

Method	BU	UA	Targeted ASR ↓	p50
No defense	67.2%	41.6%	38.9%	—
Tool Filter	73.4%	59.7%	7.5%	287 ms
Composite stack	70.6%	63.1%	5.9%	54 ms
OPA v0.58.0	69.8%	61.7%	6.8%	66 ms
GRAPHPERMIT	71.1%	67.5%	1.8%	0.95 ms

GRAPHPERMIT reduces targeted ASR by 4.1 points relative to the composite stack and by 5.7 points relative to Tool Filter while attaining 71.1% BU and the strongest UA. The policy mapping is fixed before transfer outcomes and the evaluator remains benchmark native, so this aggregate tests the representation under independently authored task semantics (Debenedetti et al., 2024).

Table 18: Policy and label sensitivity over the original workload and five reruns. Five fixed masks remove noncritical source/sink labels or widen allowlists at each level; entries report SecFN / benign success in percent.

Perturbation	Level	Composite stack	OPA v0.58.0	GRAPHPERMIT
Clean	0%	7.4 / 93.8	6.7 / 93.0	0.9 / 96.6
Label removal	5%	8.5 / 93.5	7.8 / 92.8	1.6 / 96.2
	10%	10.1 / 93.0	9.4 / 92.3	2.8 / 95.7
	20%	13.9 / 91.9	12.8 / 91.2	5.7 / 94.8
Allowlist widening	5%	8.0 / 94.2	7.2 / 93.4	1.2 / 96.9
	10%	9.1 / 94.7	8.0 / 93.9	1.8 / 97.1
	20%	11.7 / 95.1	10.2 / 94.4	3.5 / 97.5

Missing labels weaken every method, while allowlist widening trades higher benign completion for security. GRAPHPERMIT retains the largest compositional advantage at all perturbation levels. Its policy-induced false-reject rate under 5%, 10%, and 20% label removal is 1.8%, 2.3%, and 3.2%; witnesses identify the missing source or sink label for every safe rejection.

Benign success distribution.

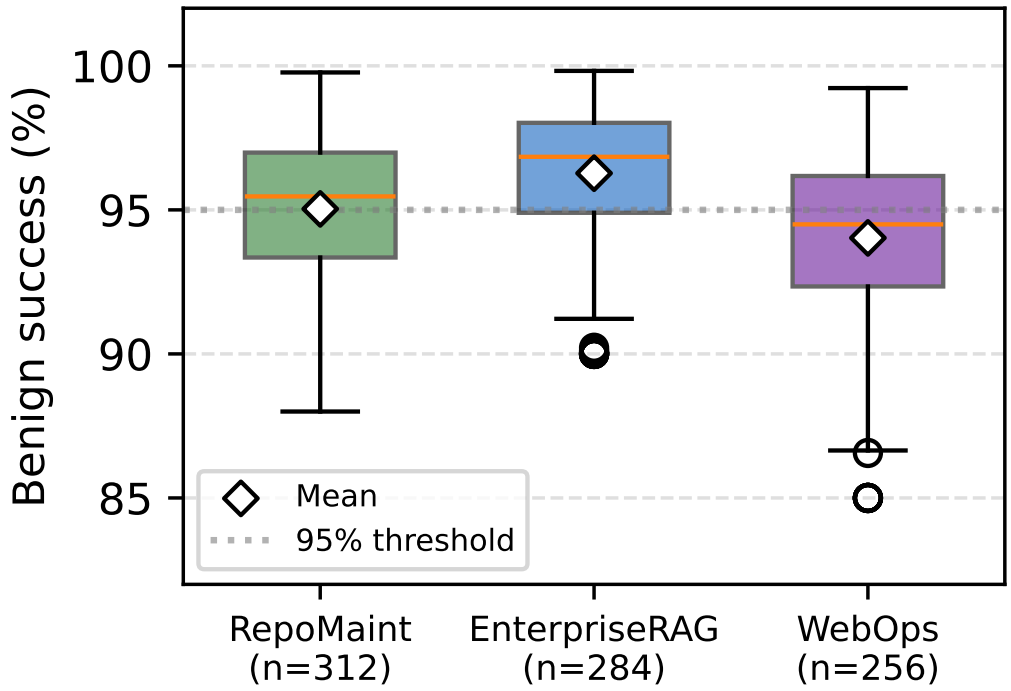


Figure 4: Benign task success distribution for RepoMaint ($n = 312$), EnterpriseRAG ($n = 284$), and WebOps ($n = 256$); $n = 852$ total. Boxes show interquartile range and diamonds mark means.

Benign medians remain high in all three deployments, so the aggregate is not carried by one setting. Failures concentrate in a comparatively small task tail rather than appearing uniformly, localizing the workflows where provenance rules and capability schemas require tuning.

Policy hardening frontier.

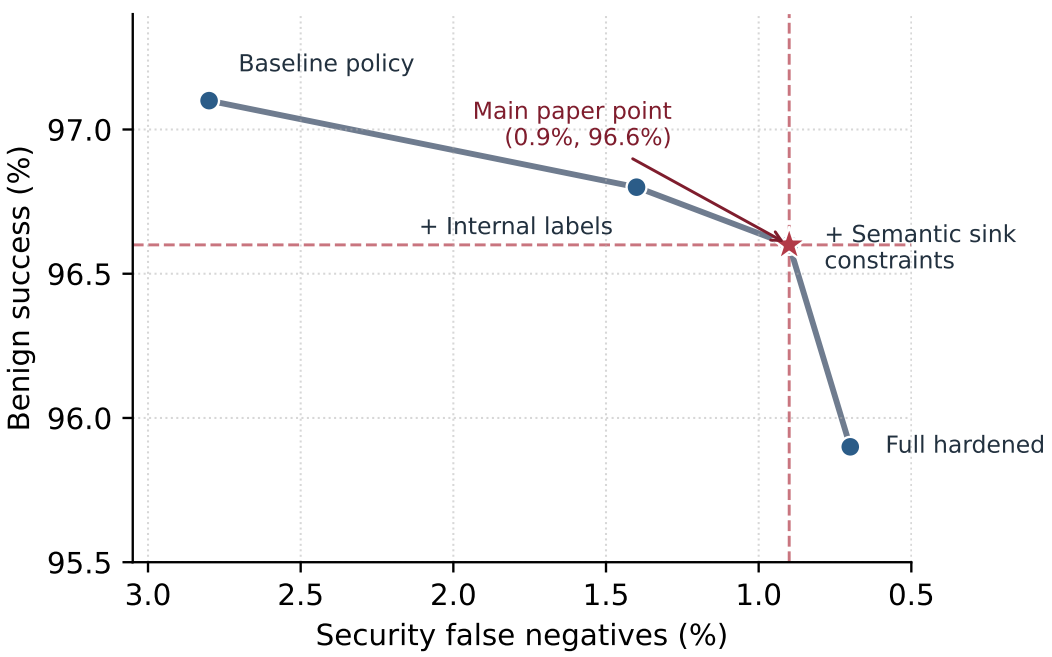


Figure 5: Policy-hardening frontier between security false negatives and benign success across progressively stricter policy settings. The starred point marks the main-paper operating point reused by Tables 3–9 (0.9% security false negatives, 96.6% benign success); the remaining points are appendix-only settings from the same hardening sweep.

Figure 5 exposes the tradeoff behind Table 3's selected operating point. The starred 0.9% / 96.6% point lies near the frontier elbow and is reused throughout the main tables. More aggressive points reduce security misses by tightening provenance and sink rules while increasing false rejects. The sweep therefore separates the boundary intervention from policy aggressiveness and shows how both residual misses and benign utility move under operator tuning.

Multi-model robustness.

Table 19: Security false negatives across models and temperatures on WebOps. Each cell contains all 206 scenarios with five reruns ($n = 1{,}030$ runs). The policy, typed gateway, and scoring rule are fixed across cells.

	GraphPermit Sec. FN (%)			Undefended Sec. FN (%)		
Model	T=0	T=0.7	T=1.0	T=0	T=0.7	T=1.0
Llama-3.1-70B	0.7	0.8	0.9	28.4	34.2	41.6
GPT-4o	0.6	0.7	0.8	22.1	29.8	38.3
Claude-3.5-Sonnet	0.5	0.6	0.7	19.6	26.4	35.1

The defended rate remains between 0.5% and 0.9% across model and temperature changes because the gate reasons over the typed plan rather than model-specific token behavior. This WebOps slice reports workload SecFN, whereas Table 3 reports its attack-blocking complement over the three-setting workload. The contrast shows that the authorization boundary is substantially less sensitive to planner variation than the unmediated agent.

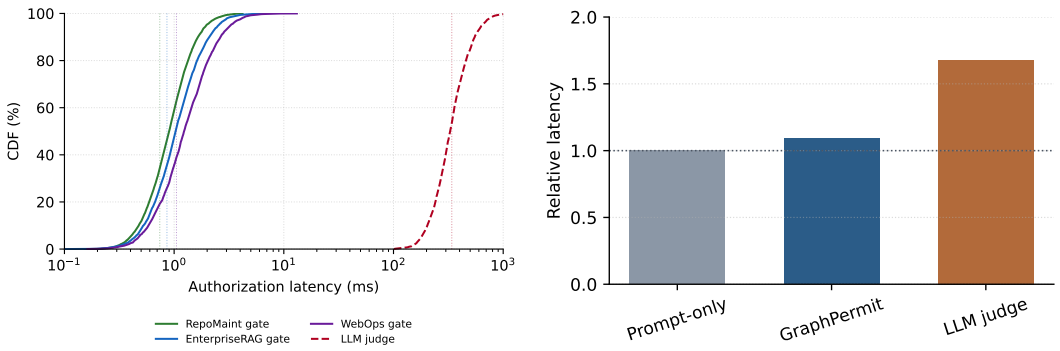
Microbenchmarks: gate latency and witness size.

Table 20: Microbenchmarks over 10,847 candidate plans compiled across all settings. Gate latency excludes inference and tool execution; witness size counts nodes in the minimal rejection subgraph.

Metric	p50	p90	p99	max
Gate latency (ms)	0.87	1.42	2.31	4.18
Graph nodes $\|V\|$	38	72	118	184
Certificate nodes	5	9	16	24

The 10,847-plan count exceeds the 7,350 benign-task and attack reruns because it includes retries, safe rejections, and state-dependent replans, which contribute 3,497 additional compilations. Linear propagation keeps p99 gate time at 2.31 ms across graphs reaching 184 nodes. Rejection witnesses remain much smaller than their source graphs (five nodes at the median and 16 at p99), keeping decision replay concise across the observed size range.

Gate latency and end-to-end overhead.



(a) Gate latency CDF.

(b) End-to-end overhead.

Figure 6: Latency diagnostics. The gate remains sub-5 ms at p99, and the dominant end-to-end overhead comes from graph construction and propagation rather than from the final policy decision itself.

Figure 6 separates deterministic gate time from the surrounding system cost. Gate latency remains tightly concentrated even in WebOps, while the judge baseline is orders of magnitude slower. Graph construction and propagation account for most of the 1.09× end-to-end overhead, identifying the relevant optimization path without conflating it with the final accept/reject branch.

Failure localization at two operating points. At the 0.9% main operating point, the 25 benign false rejects divide into 9 strict-provenance decisions, 7 capability mismatches, 5 argument-predicate failures, and 4 conservative-taint decisions. These machine-assigned rule categories identify which policy primitive produced each rejection.

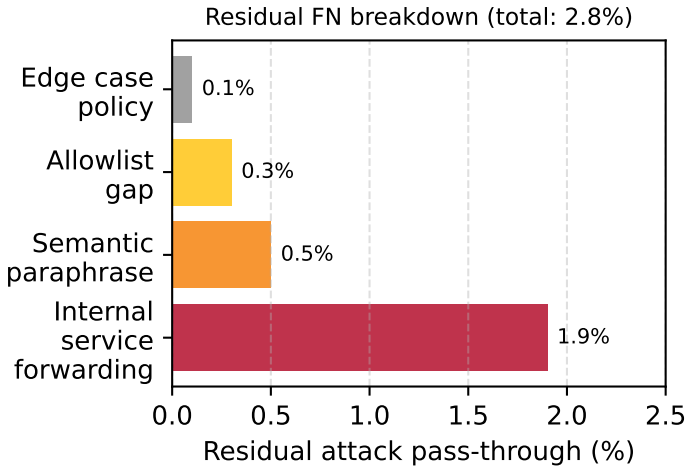


Figure 7: Residual-attack mix at the separate 2.8% SecFN frontier point. This diagnostic point provides a larger failure set than the 0.9% main operating point and is not used for headline results.

At the alternate point, residual misses cluster around internal forwarding and semantic paraphrase. Keeping its denominator separate from the main-point false-reject counts distinguishes policy-governance failures from representation and tool-semantics failures.

Attack-family distribution.

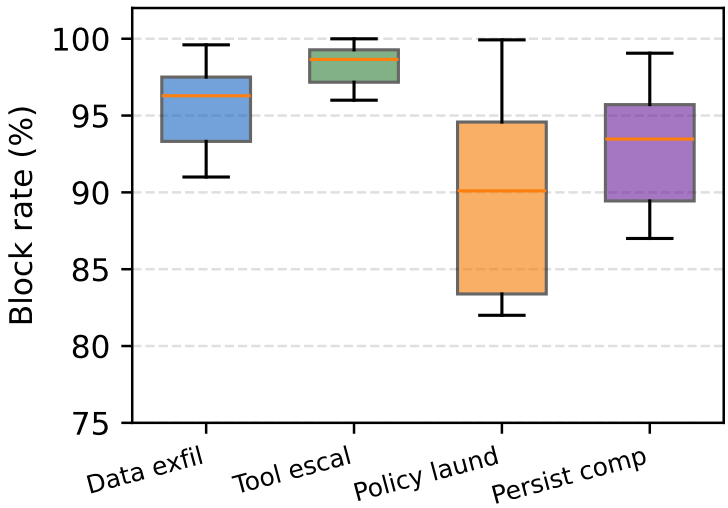


Figure 8: Per-scenario blocking-rate variation across the 206 development attacks, grouped by family. The upstream-dependent laundering and persistence families contain the harder request-local tail.

The distribution complements the family means in Table 2: direct exfiltration and escalation concentrate near the high-blocking end, whereas laundering and persistence contain more low-blocking request-local cases. This is the predicted location of information loss under terminal projection.

Parser robustness.

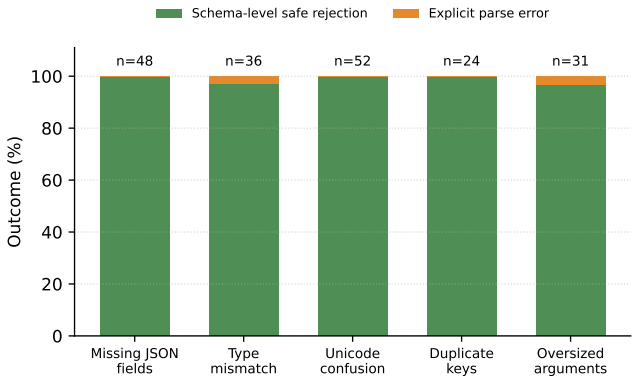


Figure 9: Malformed-input fail-closed robustness on 191 adversarial tool-call inputs across five failure families. Every case terminates as an explicit parse error or schema-level safe rejection before gateway execution; the valid-execution audit above separately measures node, dependency, and decision fidelity.

Figure 9 measures the malformed-input half of extraction robustness: ambiguous calls terminate before gateway execution. The valid-trace audit measures the complementary property on well-formed executions, including decisive-edge recall and downstream decision impact. Together they quantify both fail-closed behavior and dependency preservation at the trusted parsing boundary.