ADMISSION CONTROL FOR MULTI-AGENT LLM SYSTEMS: ENFORCING SAFETY INVARIANTS ON THE CONNECTION GRAPH

Anonymous authors
Paper under double-blind review

ABSTRACT

Agents in a multi-agent language model system now connect and delegate to one another at run time, with no human approving each link. The danger is spread along a path: one agent reads a confidential file, a second summarizes it, a third mails it out, and every step is permitted. Defenses today watch this graph without controlling it: a learned monitor covers only graphs like the ones it was trained on, and design-time advice cannot stop agents that rewire themselves as they run. We introduce TopoGate, which controls the graph instead of watching it. Before any connection is added, TopoGate checks whether it would complete a forbidden path and refuses it if so, using a fixed rule with nothing learned in it; a model is consulted only at the declassifier and overseer positions. Checking one connection at a time raises a worry: a later connection might undo an earlier decision. We prove that a gate admitting one change at a time needs every rule to be insertion-monotone, so that no later connection can make an unsafe graph look safe, and that writing each rule as the absence of a forbidden structure guarantees this and makes the gate sound over any sequence of changes. This is easy to get wrong: our own earlier rule for bounding commitment chains failed the condition, and one unrelated edge was enough to exploit the failure. In experiments, a trained graph monitor matches TopoGate on its own training graphs and loses almost all of its effect once the topology changes shape, while TopoGate holds. TopoGate also stops leaks that a text scanner misses, by blocking the outgoing connection of any agent that has read confidential data, whatever that agent's message says. Routing legitimate traffic through a declassifier node restores full task performance on benign work. The evidence is strongest for exfiltration; the human-oversight rule is tested only on a synthetic multi-agent substrate.

1 INTRODUCTION

A finance agent reads a client file, a research agent summarizes what it was given, and a reporting agent sends the summary to an address that arrived in the request. No individual step is alarming and no model refused anything it should have refused, yet the composition is an exfiltration. Multi-agent language model systems now send mail, move money, and delegate to one another without a human reading every step (Hammond et al., 2025), and models placed under goal conflict will construct such compositions deliberately (Lynch et al., 2025). The failures that matter in this setting are properties of a *path*.

Three families of defense already work at that granularity. *Learned graph monitors*, chiefly G-Safeguard (Wang et al., 2025), fit an anomaly score over the agent utterance graph and prune edges from agents the score flags. *Topological design advice*, chiefly NetSafe (Yu et al., 2025), measures which static shapes resist adversarial spread and recommends building the safer ones. *Enforcement by construction* replaces the score with a deterministic refusal, labeling values (Debenedetti et al., 2025; Siddiqui et al., 2024), constraining tool calls (Shi et al., 2025), or filtering messages (Abdelnabi et al., 2025); Appendix B surveys the family in full. The strongest published attack is itself flow-theoretic (Khan et al., 2025), searching the communication topology for a route, which makes it the right adversary for a defense that removes routes by construction.

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

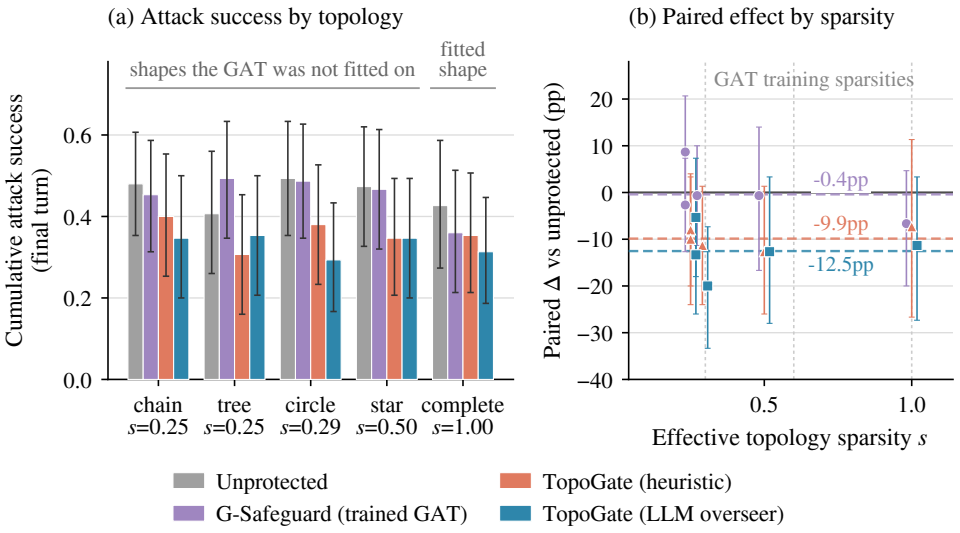

Figure 1: A trained monitor protects only where it was trained, while TopoGate holds across topology shape, so the same defense receives opposite verdicts on two distributions (Section 4.1). (a) Final-turn cumulative attack success across five canonical topologies ordered by sparsity; error bars are case-clustered bootstrap 95% intervals over $n = 30$ cases per topology. (b) Paired change against the unprotected system, dashed lines marking each defense's aggregate effect.

We take the enforcement-by-construction stance and apply it to an object that family leaves untouched: the connection graph itself. Those mechanisms decide what a value may contain, which tool an agent may call, or what a message may carry, and none decides whether a new *connection* may exist; once agents wire themselves together at run time, the only defenses that still see the graph are the learned monitor and the design-time advice, and both merely watch. Watching fails in two ways. A monitor fitted to graphs of one shape loses its protection when the system, or an attacker, reshapes the graph, which we call *distributional expiry*; design-time advice has no training distribution, but nothing enforces it once agents rewire, and the shape it recommends is expensive, with NetSafe reporting a 29.7% drop in task performance on its safest topology. And any defense that checks changes one at a time, ours included, has committed to its earlier answers, so a later change can turn an earlier admission into a mistake, which we call *sequential expiry*.

TopoGate is built to survive both. It checks every proposed change to the connection graph, a new inter-agent link, a delegation, or a commitment propagation, and refuses the change when it would complete a forbidden path. The properties it protects are graph *invariants*, deterministic predicates with nothing fitted in them, so like a stateful firewall rejecting any flow that violates a reachability policy, they have no training distribution to leave. Their violations persist under later change only under the condition Section 3 makes exact, insertion-monotonicity, and the condition is easy to miss: our own earlier rule for bounding commitment chains failed it, and one unrelated edge was enough to exploit the failure.

Our contributions are threefold. (i) **A condition for locally issued verdicts, and its reach:** for serialized admission over arbitrary edit sequences, insertion-monotonicity of every enforced predicate is necessary for soundness and the absence-of-structure form is sufficient, with our retired predicate as the necessity witness; a promote-only condition on labels follows one level down, and neither condition is specific to graphs (Section 3). (ii) **An admission gate instantiating the condition:** four safety properties become predicates over typed capability-endpoint nodes, three forbidden outright and one only bounded, and an incremental algorithm decides I_{couple} at a per-edit cost proportional to the affected subgraph (Section 2). (iii) **Measured distributional expiry:** a trained monitor and TopoGate agree in distribution and separate by 12.1 points outside it, position-based taint blocks exfiltrations that content inspection admits, and each suppression result is preceded by *elicitation*, a measurement that the failure occurs without the defense (Section 4).

2 TOPOGATE

Three things can go wrong with the agents of Section 1: the summary is mailed outside the company, the finance agent trades on it before any person sees it, or the trade sets off a chain of commitments that runs on without re-authorization. On the connection graph each is a path of a forbidden kind, and TopoGate refuses any new connection that would complete one.

2.1 THE GRAPH THE GATE PROTECTS

An agent is the wrong unit for these paths: the reporting agent both reads and writes, and exfiltration is exactly the relation between those two acts, so TopoGate expands each agent into typed sub-nodes, one per capability it exercises.

The result is a directed *guarded graph* $G = (V, E)$ whose nodes are capability endpoints. Each node carries a type $\tau(v)$ drawn from `read`, `compute`, `write`, `declassifier`, `human`, and `commit`, a sensitivity label $\sigma(v)$ on reads, a destination label $\delta(v)$ on writes, a high-impact flag, and an optional coalition tag (Table 3, Appendix A). Edges are permitted information flows: an edge from u to v asserts that content available at u may reach v.

Agents propose new connections while the system runs, and we call each proposed change a *graph edit* e, inserting a node or an edge, and write $G \sqsubseteq G'$ when G' is reachable from G by a sequence of insertions. Insertions are the only edits handled incrementally; a deletion or edge-kind change routes to full recomputation, since growth-only change is what makes the incremental guarantee provable (Section 3).

2.2 SAFETY PROPERTIES AS GRAPH PREDICATES

Written over this graph, the three failures become *predicates*, true or false with no threshold or score, which we also call *invariants* because the gate keeps them true as the graph grows.

Definition 1 (Coupling, dominance, and propagation). *Let* $S = \{v : \sigma(v) = \texttt{sensitive}\}$, $X = \{v : \delta(v) \in \texttt{external}\}$, $H = \{v : \tau(v) = \texttt{human}\}$ *and* $D = \{v : \tau(v) = \texttt{declassifier}\}$.

- $I_{\text{couple}}(G)$ *holds if no node in* S *reaches a node in* X *in* $G \setminus D$.
- $I_{\text{dom}}(G)$ *holds if every path from any source, a node in* $R = \{v : \tau(v) = \texttt{read}\}$*, to any high-impact sink contains a node of* H.
- $I_{\text{ttl}}(G)$ *holds if the longest path in the subgraph of* `commit`*-kind edges is at most* k*, where any cycle in that subgraph counts as unbounded.*

Each names a documented multi-agent failure mode (Hammond et al., 2025), exfiltration, oversight, and escalation, and all three state the absence of a path of a fixed kind, which Section 3 shows makes them safe to enforce one edit at a time.

2.3 WHICH PROPERTIES CAN BE FORBIDDEN OUTRIGHT

A ban applies to every instance of a structure, including those the system needs: blocking all confidential-to-external flow stops the exfiltration of our opening scene and also an analyst emailing a colleague a summary, since the two trace the same path and have the same degree. We therefore call a property *hard*, and enforce it by construction, only when legitimate traffic has a sanctioned route that does the same work.

All three predicates admit one: forwarding through a declassifier (Section 2.5), acting through a human node, and re-authorizing a long commitment chain. TopoGate therefore enforces $\Phi = I_{\text{couple}} \wedge I_{\text{dom}} \wedge I_{\text{ttl}}$.

A fourth candidate admits none. $I_{\text{cut}}(G)$ bounds the maximum flow from any coalition to any sink (Edmonds-Karp on unit-capacity edges), but the structure it forbids covers collusion and ordinary cooperation alike, so it can only cap bandwidth and hand the residual to a *soft* layer that judges where the hard predicates decide by structure.

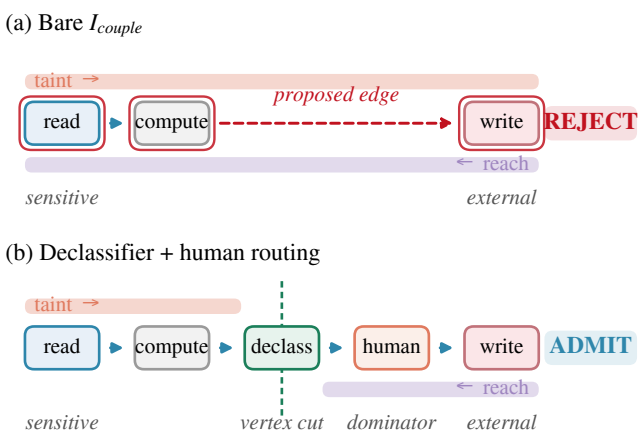

Figure 2: Admission is decided on graph structure; the payload is identical in both panels. A node carrying both taint (forward from a sensitive read) and reach (backward from an external write) witnesses a sensitive-to-external path. (a) The proposed edge makes the bands overlap, so the edit is rejected. (b) The declassifier is a vertex cut, the bands stay disjoint, and the same forward is admitted through a human node that satisfies I_{dom}.

2.4 Checking Φ at inference time

TopoGate exposes a single interface, $\textsc{check}(G, e) \to \{\textsc{admit}, \textsc{reject}\}$. On a proposed edit it evaluates each hard predicate on the trial graph $G \cup \{e\}$, consults the soft overseer if the edit touches a high-impact path, and admits if and only if every hard predicate holds and the overseer does not block. Adding a node alone cannot violate a flow predicate, since only an edge can create a new path. All agents' proposals pass through this one check-and-commit point, which evaluates an edit against the committed graph and commits it before the next is checked; a concurrent deployment must preserve this serialization, since two edits checked against the same snapshot can each pass and jointly violate Φ.

Recomputing reachability from S to X after every edit costs $O(VE)$, too slow to run before each connection commits, so TopoGate instead caches two bits per node: $\texttt{taint}(v)$, true when v is reachable from a sensitive read while bypassing declassifiers, and $\texttt{reach}(v)$, true when v can reach an external write while bypassing declassifiers. A node holding both bits is itself a witness to a sensitive-to-external path, so a global path question becomes a local per-node test. On inserting (u, v), a forward search from v propagates taint and a backward search from u propagates reach, both halting at declassifiers, and the edit is rejected if any node ends with both bits set (Algorithm 1, Appendix A).

Two consequences matter later. Per-edit cost of maintaining the bits is proportional to $|K_1|$, the affected subgraph, plus the edges incident to it, rather than to $|V|$ or $|E|$; I_{dom} and I_{ttl} are evaluated directly on the trial graph, and Section 4 measures whether $|K_1|$ stays small on real traces. And because both bits are least fixpoints over a graph that only grows, an insertion can set a bit but never clear one, which Section 3 shows is exactly what a gate needs to stay sound across a sequence of edits.

2.5 Recovering the utility a refusal costs

As Section 2.3 showed, I_{couple} blocks a legitimate forward as firmly as an exfiltration. The resolution gives the legitimate forward a route the exfiltration cannot take. A declassifier node is a vertex cut, so taint stops there, and a forward routed $\texttt{read} \to \texttt{declassifier} \to \texttt{human} \to \texttt{write}$ satisfies I_{couple} structurally and satisfies I_{dom} by passing a human (Figure 2). Exfiltration gains nothing from the route, because the declassifier withholds it, so residual risk concentrates on one auditable chokepoint, which any of three interchangeable backends can occupy: the structural guarantee is

unchanged by the choice and only the residual-risk term moves (Appendix E.2). Our experiments use the model-based backend, with data-dependency edges reconstructed by the provenance tracker of Appendix E.3.

One limit follows from the object being protected: influence carrying no explicit edge, such as a shared context window or steganography over message content, is beyond any reachability predicate over G and falls to I_{cut} or to a per-channel layer such as value-level labeling (Debenedetti et al., 2025) (Appendix F.3). On AgentDojo's workspace suite, 31 of 48 causal dependencies between tool calls are edges the gate mediates (44 to 65% depending on detection strictness), and all 3 carried through shared environment state, such as a file one call writes and another reads, are missed (Appendix C); as shared scratchpads and filesystems are common in multi-agent systems, the guarantee covers only flows that surface as edges.

3 When a Verdict Survives Later Edits

A verdict issued on each edit in isolation is exposed to sequential expiry: individually admitted edits might assemble a violating graph. This section gives the condition that rules this out and shows that neither it nor its counterpart on labels comes for free; proofs are in Appendix D.1.

3.1 Insertion-monotonicity

A gate that cannot revisit its earlier answers needs violations that later edits cannot undo.

Definition 2 (Insertion-monotone violation)**.** *A predicate I is* insertion-monotone *if its violation persists under insertion: whenever G violates I and $G \sqsubseteq G'$, then G' violates I.*

Monotonicity looks like a condition a designer has to impose, but any correct encoding of a path property already obeys it.

Lemma 1 (Existential structural properties are monotone)**.** *Let $I(G) = \neg\,\exists\,\pi \in \Pi(G)$, where $\Pi(G)$ is the set of substructures of a fixed shape: a path between two label-determined vertex sets, a path of length exceeding k, a cycle, or a flow of value exceeding a cap. If the vertex sets are fixed and $G \sqsubseteq G'$, then $\Pi(G) \subseteq \Pi(G')$, so I is insertion-monotone.*

This discharges all four predicates in one step, since each is violated exactly when a substructure of the required shape exists: an $S \to X$ path in $G \setminus D$ for I_{couple}, a source-to-sink path avoiding H for I_{dom}, an over-length commit path or commit cycle for I_{ttl}, and an over-cap flow for I_{cut}. Every vertex set these shapes refer to, S, X, H, D, the sources R and the high-impact sinks, is determined by node labels alone, so none shrinks under insertion. I_{cut} is thus monotone and is excluded from Φ only because it lacks a sanctioned route (Section 2.3): monotonicity decides whether a property *can* be enforced one edit at a time, and a sanctioned route whether it *should* be forbidden outright.

3.2 Sequence soundness, in both directions

Because the gate is edit-local, soundness first rests on the cached summary being an exact account of the graph.

Theorem 1 (Incremental adequacy)**.** *Let* taint *and* reach *be the least fixpoints of the propagation operators of Section 2.4. After every insertion in a sequence, the maintained bits equal the bits recomputed from scratch on the same graph, and the work performed is $\Theta(|K_1|)$ plus the edges incident to K_1, where K_1 is the set of nodes whose bits changed.*

The proof is a standard fixpoint argument whose substance is excluding deletion, which can shrink the least fixpoint and leave the previous one an unsound seed. Given adequacy, the remaining question is whether a chain of local verdicts composes.

Theorem 2 (Compositional sequence soundness)**.** *Let P be an intended property of existential structural form, and let an edit-local gate that admits edits one at a time through a single check-and-commit point enforce a predicate Φ raising no false alarms, $\neg\Phi(G) \Rightarrow \neg P(G)$ for every G. Let G_0 satisfy Φ. (i)* Sufficiency. *If Φ has the existential form of Lemma 1 and $\Phi \Rightarrow P$, every sequence of edits each admitted in turn yields a graph satisfying P. (ii)* Necessity. *If Φ is not insertion-monotone, there are $G_a \sqsubseteq G_b$ with $\neg\Phi(G_a)$, $\Phi(G_b)$ and $\neg P(G_b)$, so Φ does not imply P. If*

moreover $G_0 \sqsubseteq G_b$ and $E(G_b) \setminus E(G_0)$ has an enumeration every prefix of which satisfies Φ, the gate admits every edit of that enumeration and ends in a graph violating P.

Sufficiency follows from Lemma 1 and an induction along the sequence. Necessity is the direction that constrains a designer: a non-monotone predicate that raises no false alarms necessarily misses a violation, and whenever the missed graph can be reached through admitted prefixes, the gate admits a sequence whose every edit is accepted and whose final graph is unsafe. The no-false-alarm condition is what gives necessity content: a predicate that also rejects some safe graphs, such as $P \wedge (|V| \neq 7)$, can be non-monotone and still sound. That claim requires a witness, and Section 3.3 supplies one from our own system.

3.3 The necessity witness

Lemma 1 covers existential predicates, but one written in universal form over a set the labeling induces need not be monotone at all. If $I(G) = \forall\, v \in \mathcal{D}(G).\ \varphi(G, v)$ and $\mathcal{D}$ can lose elements as G grows, an insertion that removes from $\mathcal{D}$ every witness to $\neg\varphi$ moves the verdict from violating to satisfied.

An earlier formulation of ours had exactly this shape, and it shipped: it bounded the longest commit path from each origin, with $\mathrm{Origins}(G) = \{v : v \text{ has no incoming commit edge}\}$, a set that insertion shrinks. A five-edit sequence builds a commit cycle and then supplies an unrelated origin elsewhere; the cycle guard stops firing, the cyclic component is never scanned, and the predicate reports safety on a graph whose commit chain is infinite. Every edit is admitted, which is the witness Theorem 2 requires; Appendix D.2 gives the sequence, executed against the shipped gate.

Counting any commit cycle as unbounded, as Definition 1 does, returns the predicate to existential form and restores the theorem. The design rule is mechanical: an enforced predicate must be expressible as the negation of an existential structural property, and universal quantification over any domain that insertion can shrink is forbidden.

3.4 Labels are the second condition

Theorem 2 holds over a fixed labeling, and that assumption hides a second way for a verdict to expire: the predicates read labels to decide which nodes are sensitive and which external, so a gate can enforce a monotone predicate flawlessly and still be unsound if those labels are wrong.

An audit of calendar tools found a write node statically labeled internal whose destination becomes external when an out-of-organization participant arrives as a tool *argument*. Three admitted edits then leave a graph on which the gate reports that I_{couple} holds while the labels the tools actually realize violate it. Checking all three atomically does not help, and no adversary is needed: an unpatched static-label gate admitted such sequences in 499 of $15{,}000$ clean-start runs.

The repair is to compute labels from arguments rather than read them off the node, and whether it suffices turns on a sharp condition. Order labels by *exposure*, writing $\ell \sqsubseteq_{\mathrm{exp}} \ell'$ when ℓ' is at least as exposed in every component, and let a node's effective label be the join of its static label with the labels of all arguments on incoming edges (Appendix D.4).

Proposition 1 (Label-monotonicity condition)**.** *The argument-aware gate is sequence-sound over effective labels* ***if and only if*** *the argument-to-label map is* promote-only*: every rule is monotone into $\sqsubseteq_{\mathrm{exp}}$ and can move a label only toward greater exposure.*

Arguments may therefore widen a node's exposure but never narrow it on the incremental path. A genuine need to narrow must originate at a node the gate already trusts to clear taint, or else route to full recomputation; our implementation machine-checks this (Appendix D.4). Both conditions say that a locally checked defense is sound exactly when later events cannot retract its verdicts, whether through the predicate or through the labels it reads.

3.5 How far the conditions reach

Neither condition mentions graphs, admission, or TopoGate, so both bind every member of the enforcement family of Section 1. Read from their published descriptions, two of those systems have

Table 1: Attack success on two distributions, as final-turn cumulative incidence. The in-distribution column uses the random-sparse trajectories G-Safeguard was fitted on; the second pools the five canonical NetSafe shapes, of which complete ($s = 1.0$) coincides with a training sparsity and is kept in the pool, which favors G-Safeguard. Paired differences against G-Safeguard, with case-clustered bootstrap 95% intervals: TopoGate-O is indistinguishable in distribution (-4.3, $[-9.2, +0.5]$, $p = 0.079$) and separates across topologies (-12.1, $[-18.4, -5.8]$, $p = 0.0002$). Appendix E.4 compares these intervals against the slot-independent Wilson alternative.

Defense	In distribution ($n = 120$)	Five topologies ($n = 150$)
Unprotected	0.638	0.456
G-Safeguard (Wang et al., 2025)	0.412	0.452
TopoGate-H (training-free)	0.500	0.357
TopoGate-O	0.368	**0.331**

the shape Proposition 1 forbids, in the mechanism each is known for, while Progent (Shi et al., 2025) satisfies an analogue over privilege sets. Both of the former relax label propagation deliberately and argue soundness under their own threat models, so the shape means their verdicts can be retracted by later events, not that they fail the guarantee they claim. Checking the condition is mechanical, and a falsifier recovers our retired counterexample in 20 trials (Appendix D.3).

4 Experiments

Threat model. We assume an adversary that controls the content of injected instructions and, in the multi-agent setting, some subset of the agents. It knows the defense completely, including the predicate definitions, the labeling function, and the implementation, since TopoGate is deterministic and public. It may propose arbitrary graph edits and choose their order freely, but it cannot modify the gate, the label map, or the trial-graph check, and it cannot delete edges. Influence carrying no explicit graph edge lies outside what the gate observes. We follow the flow-theoretic adversary of Khan et al. (2025) without running their released optimizer, and Appendix E.1 states what that bounds.

Benchmarks. Four sources supply the evaluation. The multi-agent trajectories come from G-Safeguard's tool-attack harness, and the topology sweep varies their adjacency over the five canonical NetSafe shapes. InjecAgent (Zhan et al., 2024) provides two-step exfiltration cases, whose attack shape is the one I_{couple} names. AgentDojo's workspace suite (Debenedetti et al., 2024) measures benign utility. Case counts, sparsity values, model identities, and tool inventories are in Appendix E.1.

Arms and baselines. Every number is labeled by one of two TopoGate arms. **TopoGate-H** decides by a deterministic content rule and uses no model at all; **TopoGate-O** consults a model overseer at the chokepoint. Only TopoGate-H is training-free end to end. Baselines are the unprotected system, a NetSafe-style chain constraint rejecting any edge that raises a node's degree above one, a content-inspection sanitizer scanning outbound payloads for sensitive strings, and the trained graph attention monitor of Wang et al. (2025), which we trained on G-Safeguard's own training split. Both defended arms act through G-Safeguard's per-agent demotion, so they differ only in what decides to flag, and attack success uses G-Safeguard's published criterion unmodified. The head-to-head therefore compares a structural predicate with a learned score as the flagging signal; it does not compare edge-level admission with per-agent pruning, which would require varying the intervention as well and remains the largest gap in our evaluation.

4.1 A learned monitor's protection expires off its training distribution

The first question is whether a training-free predicate can match a trained monitor where the monitor is strongest. Table 1 shows that it can: on the trajectories G-Safeguard was fitted on, the two are statistically indistinguishable.

The separation appears as soon as the graph shape moves off that distribution, with only the adjacency generator changed. Across the five canonical shapes G-Safeguard's effect against the unprotected system falls within noise of zero, $[-6.0, +5.1]$, while TopoGate-O removes 12.5 points,

Table 2: Exfiltration on InjecAgent ($n = 544$ per injection setting), a benchmark whose attack shape is the one I_{couple} names. Counts in parentheses; Wilson 95% intervals put the upper bound on every 0.00% cell at 0.70%. The right-hand columns condition on the cases that reach the external write, which is the only stage at which the two taint rules can differ. The sanitizer acts on content, not at a graph position, so it has no count there.

	Attack success, all cases		Blocked at external write	
Condition	Base	Enhanced	Blocked	Attempts
No defense	5.15% (28)	7.17% (39)	0	63
Output sanitizer	0.00% (0)	0.00% (0)	—	—
TopoGate, payload taint	2.21% (12)	3.31% (18)	33	63
TopoGate, position taint	**0.00%** (0)	**0.00%** (0)	**64**	64

$[-18.7, -6.3]$, and the paired gap between the two defenses widens to $p = 0.0002$. A monitor fitted to one family of graph shapes carries no commitment to any other, so which verdict a study reports is decided by where it evaluates. Two qualifications apply. The regimes differ in more than shape, since unprotected attack success falls from 0.638 to 0.456, so part of the change may reflect easier attacks rather than topology alone. And a G-Safeguard retrained on mixed topologies is the natural remedy for this expiry; we did not test it, so the result shows that a monitor's protection is bound to its training shapes, not that retraining cannot recover it.

The training-free arm reverses across the same boundary: TopoGate-H is worse than G-Safeguard in distribution and better across topologies, at $p = 0.0018$ and $p = 0.0014$. We read this reversal as support for the aggregate separation rather than a standalone claim, and the per-shape intervals at $n = 30$ in Figure 1 are too wide to read individual cells.

4.2 Position-based taint survives paraphrase

A check that reads the outgoing message can be defeated by paraphrase, since the attacker writes the message; a check that reads the sender's position in the graph cannot. We run one mechanism two ways that differ only in this: *payload taint* asks at the external write what the message contains, while *position taint* taints any agent that used a tool the threat model designates as attacker-influenced and refuses its external write whatever the message says. The lattice machinery is standard (Denning, 1976; Myers & Liskov, 1997) and already applied to agents (Zhong et al., 2025; Siddiqui et al., 2024); the ablation is what we contribute.

Since the agent itself refuses at the first step in 412 of 544 enhanced cases, the informative comparison is the right-hand columns of Table 2: among cases reaching the external write, payload taint blocks 33 of 63 attempts and position taint 64 of 64. The gap is significant at $p < 0.05$ in both settings, and falls exactly where content inspection has nothing to inspect, since the payload paraphrases the sensitive value instead of quoting it. The sanitizer also reaches 0.00%, but by flagging sensitive-looking patterns anywhere in the call, including the destination address. Neither its benign cost nor that of position taint is measured here, and position taint's 64 of 64 is structural by construction, since every case is a read followed by a write; the result shows that position taint survives paraphrase, not what it costs on benign traffic.

Taint sources come from the threat model rather than tool names, which the adversary can choose (Appendix E.1); Figure 3 shows where each defense acts.

4.3 What enforcement costs on benign work

Since a defense that blocks everything blocks every attack, we measure what refusals cost. AgentDojo's workspace suite supplies 14 injection tasks, of which three construct a confidential-to-external flow. The remaining eleven build no such flow and are outside what I_{couple} addresses, so we report against the three rather than claiming credit for fourteen. Both configurations block all three exfiltrations, and they differ on the 40 benign tasks, whose trajectories are canonical ground truth, so every benign failure is attributable to the gate (Table 7, Appendix G.1).

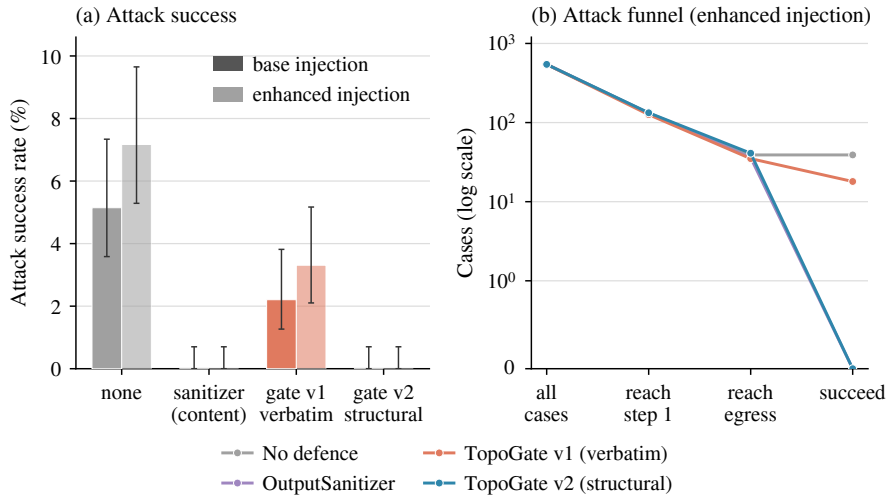

Figure 3: Position-based taint blocks the exfiltrations payload inspection admits. (a) Attack success by defence and injection setting, Wilson 95% intervals at $n = 544$. (b) The funnel to successful exfiltration under enhanced injection; the defences differ only on the cases that reach the external write.

The bare predicate blocks 7 of 40, 17.5% of benign trajectories, above the 15% ceiling we fixed before running the experiment. This is the predicate behaving as specified, since a legitimate forward and an exfiltration trace the same path. Declassifier routing removes the cost, blocking 0 of 40 while preserving the block, by moving the decision on the seven legitimate forwards to the declassifier node rather than removing it. In these experiments that node runs the model-based backend (Appendix E.2), so the utility recovery is not training-free, and the declassifier's false-permit rate against content crafted to mimic a legitimate forward is unmeasured; the 3 of 3 exfiltration block covers only the genuine cases. Appendix G.1 tabulates these counts and plots the resulting safety-utility frontier.

4.4 Further measurements

Six results are reported in full in the appendices; the last two are context rather than evidence for our claims. **Elicitation** confirms each pathology occurs without the gate before we report suppressing it; the elicitation rate is the measurement, and the post-gate zero is a conformance check, since the gate refuses exactly the structures that define each pathology (Appendix F.1). **Live replication** repeats the measurements with model calls, and both gated arms match the ungated agent's task success exactly (Appendix F.2). An **adaptive adversary** with full knowledge of the gate completed no coupling in any of 30 cells (Wilson upper bound 0.114; 6 cells ended by the agent's own refusal), across five author-written scenarios that test explicit routes, not laundering through shared state, with the mechanism split reproducing exactly across three trials (Appendix F.3). **Admission cost** stayed at a maximum of two affected nodes across 9,402 insertions (Appendix F.4). On **AgentLeak** (El Yagoubi et al., 2026), 1,000 single-hop scenarios never fire the structural predicate, and a single-trial **CrewAI** case study shows complementary coverage rather than a ranking (Appendices G.2).

5 Conclusion

Safety properties usually pursued with learned monitors can instead be stated as graph predicates and enforced by admission control at edit time, where the verdict does not expire: a predicate has no training distribution to leave, and an insertion-monotone one cannot be retracted by later serialized edits. Learned monitors remain useful, but a structural cut decides where a fallible check happens, at the price of a labeling obligation and a boundary at the implicit channel. What we expect to transfer are the two conditions, insertion-monotone predicates and promote-only labels, and one methodological point: a suppression rate of zero says nothing until the unprotected rate is measured on the same substrate.

AI USE STATEMENT

We used large language models as assistants in four roles: polishing the writing; retrieval and discovery, including finding and summarizing related work and searching for relevant papers; research ideation and planning, including discussing gaps in the research idea; and writing code, testing hypotheses and implementing additional ablation studies. The authors verified every output of these tools, and all ideas, claims, experimental results and conclusions are the authors' own responsibility.

ETHICS STATEMENT

This work develops a defense against exfiltration and unsafe action in multi-agent language model systems and involves no human subjects. All benchmarks are public, and every confidential record and recipient address in our experiments is benchmark-provided or fabricated. Live adversarial runs use an authorized red-team framing stating that the data is fabricated, solely to test the gate against an agent committed to attacking. The attack constructions we report are each paired with the repair that prevents them, and our analysis of two published systems, drawn from their descriptions, identifies a structural shape rather than a vulnerability in their deployments. The gate's guarantee is limited to serialized insertions on the explicit graph under a sound labeling (Appendix C), so it should be deployed as one layer of defense in depth.

REPRODUCIBILITY STATEMENT

Section 4 and Appendix E.1 state the benchmarks, sample sizes, models, temperatures, and scoring criterion, and Appendix F.1 gives the elicitation parameter grid and seeds. The offline tier reproduces in under five minutes on a laptop CPU with no model endpoint and is exactly reproducible across machines, because TopoGate contains no learned component on the structural path; the live tiers require a model endpoint and are seeded where the model permits. Dependencies are pinned to Python 3.11 and AgentDojo 1.2.1. Intervals are Wilson for independent cases and case-clustered bootstrap with $10{,}000$ resamples for trajectory data. All figures regenerate from released per-case score files through the scripts of Appendix E.5, and the theory claims ship with regression tests encoding both counterexamples.

REFERENCES

Sahar Abdelnabi, Amr Gomaa, Eugene Bagdasarian, Per Ola Kristensson, and Reza Shokri. Firewalls to secure dynamic LLM agentic networks. *arXiv preprint arXiv:2502.01822*, 2025. URL https://arxiv.org/abs/2502.01822.

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.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS) Datasets and Benchmarks Track*, 2024. arXiv:2406.13352.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025. URL https://arxiv.org/abs/2503.18813.

Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19 (5):236–243, 1976.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. AgentLeak: A full-stack benchmark for privacy leakage in multi-agent LLM systems. *arXiv preprint arXiv:2602.11510*, 2026. URL https://arxiv.org/abs/2602.11510.

Lewis Hammond, Alan Chan, Jesse Clifton, Jason Hoelscher-Obermaier, Akbir Khan, Euan McLean, Chandler Smith, Wolfram Barfuss, Jakob Foerster, Tomáš Gavenčiak, The Anh Han,

Edward Hughes, Vojtěch Kovařík, Jan Kulveit, Joel Z. Leibo, et al. Multi-agent risks from advanced AI. *arXiv preprint arXiv:2502.14143*, 2025. URL https://arxiv.org/abs/2502.14143. Cooperative AI Foundation, Technical Report #1.

Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

Xu He, Di Wu, Yan Zhai, and Kun Sun. SentinelAgent: Graph-based anomaly detection in multi-agent systems. *arXiv preprint arXiv:2505.24201*, 2025. URL https://arxiv.org/abs/2505.24201.

Rana Muhammad Shahroz Khan, Zhen Tan, Sukwon Yun, Charles Fleming, and Tianlong Chen. Agents under siege: Breaking pragmatic multi-agent LLM systems with optimized prompt attacks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. arXiv:2504.00218.

Peiran Li, Xinkai Zou, Zhuohang Wu, Ruifeng Li, Shuo Xing, Hanwen Zheng, Zhikai Hu, Yuping Wang, Haoxi Li, Qin Yuan, Yingmo Zhang, and Zhengzhong Tu. SAFEFLOW: A principled protocol for trustworthy and transactional autonomous agent systems. *arXiv preprint arXiv:2506.07564*, 2025. URL https://arxiv.org/abs/2506.07564.

Richard J. Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

Aengus Lynch, Benjamin Wright, Caleb Larson, Kevin K. Troy, Stuart J. Ritchie, Sören Mindermann, Ethan Perez, and Evan Hubinger. Agentic misalignment: How LLMs could be insider threats. *arXiv preprint arXiv:2510.05179*, 2025. URL https://arxiv.org/abs/2510.05179.

Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 129–142, 1997.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for LLM agents. *arXiv preprint arXiv:2504.11703*, 2025. URL https://arxiv.org/abs/2504.11703.

Shoaib Ahmed Siddiqui, Radhika Gaonkar, Boris Köpf, David Krueger, Andrew Paverd, Ahmed Salem, Shruti Tople, Lukas Wutschitz, Menglin Xia, and Santiago Zanella-Béguelin. Permissive information-flow analysis for large language models. *arXiv preprint arXiv:2410.03055*, 2024. URL https://arxiv.org/abs/2410.03055.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on LLM-based multi-agent systems. *arXiv preprint arXiv:2502.11127*, 2025. URL https://arxiv.org/abs/2502.11127.

Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. IsolateGPT: An execution isolation architecture for LLM-based agentic systems. In *Network and Distributed System Security Symposium (NDSS)*, 2025. arXiv:2403.04960.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Qingsong Wen, Kun Wang, and Yang Wang. Netsafe: Exploring the topological safety of multi-agent networks. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. arXiv:2410.15686.

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*, 2024. arXiv:2403.02691.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. RTBAS: Defending LLM agents against prompt injection and privacy leakage. *arXiv preprint arXiv:2502.08966*, 2025. URL https://arxiv.org/abs/2502.08966.

Zhenhua Zou, Zhuotao Liu, Lepeng Zhao, and Qiuyang Zhan. BlockA2A: Towards secure and verifiable agent-to-agent interoperability. *arXiv preprint arXiv:2508.01332*, 2025. URL https://arxiv.org/abs/2508.01332.

A Notation and the Admission Algorithm

This appendix collects the two reference objects the main text uses throughout. Table 3 gives the node signature and the symbols carried across Sections 2 and 3, and Algorithm 1 states the per-edit check whose cost and adequacy Sections 2.4 and 3 analyse.

Table 3: Node signature and notation.

Symbol	Meaning
$\tau(v)$	type: `read`, `compute`, `write`, `declassifier`, `human`, `commit`
$\sigma(v)$	sensitivity label, carried on `read` nodes
$\delta(v)$	destination label, carried on `write` nodes
$G \sqsubseteq G'$	G' is reachable from G by a sequence of insertions
Φ	the conjunction of predicates the gate enforces
K_1	the affected subgraph: nodes whose cached bits change on an edit

The algorithm is stated for an edge insertion, which is the load-bearing case: a node insertion cannot create a path and so cannot violate a flow predicate on its own. Rejection rolls the bits back, so a refused edit leaves the cached summary exactly as it was, which is what lets the next edit be checked against an unpolluted state.

Algorithm 1 CHECK(G, e) for an edge insertion $e = (u, v)$

$T \leftarrow \emptyset$; $R \leftarrow \emptyset$ ▷ nodes whose bits change
if `taint`(u) **and** $\tau(v) \neq$ `declassifier` **then**
  $T \leftarrow$ forward BFS from v, setting `taint`, halting at declassifiers
end if
if `reach`(v) **and** $\tau(u) \neq$ `declassifier` **then**
  $R \leftarrow$ backward BFS from u, setting `reach`, halting at declassifiers
end if
$K_1 \leftarrow T \cup R$
if $\exists\, w \in K_1 :$ `taint`$(w) \wedge$ `reach`(w) **then**
  roll back the bits and **return** REJECT
end if
return ADMIT **if** the overseer does not block, **else** REJECT

B Extended Related Work and Positioning

Learned graph monitors. G-Safeguard (Wang et al., 2025) builds a graph neural network over the agent utterance graph and prunes edges from agents a learned detector flags; SentinelAgent (He et al., 2025) extends this with temporal attributed graphs and unsupervised variants. Two properties are common to the family and are the ones we depart from: the features are surface text, and the system acts only after a learned score crosses a threshold. G-Safeguard is the closest prior system in deployment position and the strongest published baseline in this literature, which is why Section 4.1 runs against it directly rather than against a reimplementation.

Topological safety as design advice. NetSafe (Yu et al., 2025) identifies which static topological properties improve security, finding highly connected systems more vulnerable to adversarial spread. NetSafe is descriptive and static: it advises choosing safer fixed topologies and enforces nothing once agents rewire, and its central recommendation is the utility-destroying one, since a chain serializes the system. A reachability predicate can forbid the unsafe path while keeping useful connectivity; a global degree cap cannot express the distinction.

Topological attacks. (Khan et al., 2025) formulate multi-agent attack as maximum-flow minimum-cost optimization over the communication topology, distributing an adversarial prompt across bandwidth-constrained links to bypass distributed safety mechanisms. That the strongest published attack is flow-theoretic makes it the right adversary for a flow-theoretic defense: the attack searches the flow network for a route, and the predicate removes the unsafe cut from that network by construction.

Enforcement by construction. TopoGate belongs to a substantial family that already enforces deterministically instead of scoring. CaMeL (Debenedetti et al., 2025) labels each value and enforces that no labeled flow violates policy within a single agent; permissive information-flow analysis (Siddiqui et al., 2024) and RTBAS (Zhong et al., 2025) propagate labels across model calls and tool responses; SAFEFLOW (Li et al., 2025) extends provenance labeling to multi-agent settings. Progent (Shi et al., 2025) constrains which tool calls an agent may issue under a programmable privilege policy. BlockA2A (Zou et al., 2025) enforces access-control policies on an agent interoperability protocol, halting execution and revoking permissions after compromise. IsolateGPT (Wu et al., 2025) isolates components at design time, and Abdelnabi et al. (2025) filter the messages agents exchange. Fides (Costa et al., 2025) is the closest value-level system on our own benchmark: it attaches confidentiality and integrity labels to values in an agent planner, characterizes which security properties such labels can enforce, and evaluates on AgentDojo. Its selective hiding of labeled values plays the role our declassifier plays at graph level, deciding where sensitive content may be released, and the two compose: Fides governs what a value carries, TopoGate which connections may exist.

Protection graphs. Constraining an authorization graph rather than observing it predates the agent literature. Harrison et al. (1976) showed safety undecidable for general access-matrix protection systems, and the Take-Grant model made it decidable in linear time by restricting the graph rewriting rules (Lipton & Snyder, 1977). TopoGate inherits that stance. What differs is the object, a connection graph that language model agents rewrite at run time, and the question, which is not whether a fixed rule set can ever reach an unsafe state but under what condition verdicts issued edit by edit compose (Section 3).

Section 1 states the two differences in brief, and the details are these. BlockA2A comes closest to a runtime gate, but it decides whether an agent may invoke a capability and revokes afterward, and no member of the family asks whether admitting a proposed connection would complete a forbidden reachable path, which is the question a self-rewiring graph forces. On the second difference, every member checks incrementally, as each value, call, or message arrives, and so inherits the obligation that Section 3 makes precise; Section 3.5 reports what that obligation says about them.

Table 4: Positioning against the closest prior systems, cited in the text above. The enforcement family constrains values, calls, and messages within a fixed execution; TopoGate constrains the connection graph those agents rewrite. The last column is the obligation Section 3 identifies, which applies to every incrementally checked entry. *Shape present* means the mechanism has the form Proposition 1 forbids, established by reading the published description rather than by running the system (Section 3.5).

System	Object constrained	Runtime enforcement	Keeps conn.	Incr. check	Monotonicity condition
G-Safeguard	graph	prune (learned)	yes	no	n/a
NetSafe	graph	advice only	no	no	n/a
BlockA2A	capability	revoke	yes	no	n/a
CaMeL	value	yes	partial	yes	not determined
Progent	tool call	yes	yes	yes	satisfied
RTBAS	value	yes	yes	yes	shape present
Permissive IFC	value	yes	yes	yes	shape present
TopoGate	**connection**	admission	yes	yes	proved

C LIMITATIONS IN FULL

Scope of the guarantee. The guarantee covers serialized single-edit insertion sequences under a fixed, sound labeling: every edit is checked against the committed graph and committed before the next is checked, as in our implementation, and a concurrent deployment must enforce the same order, for instance with a lock around check-and-commit. Deletions and edge-kind changes route to full recomputation. The coalition-flow predicate is a bound and is not part of the enforced conjunction. Where a legitimate sensitive-to-external forward exists, the decision rests on the declassifier or the overseer, so residual risk concentrates at that chokepoint by design, and the claim is structural containment of a fallible monitor rather than structural impossibility.

What the edge representation covers. The guarantee is relative to the graph, so influence travelling by some other route falls outside it. We can now say by how much on one suite rather than only in principle.

Method. On the workspace ground truth we detected causal dependencies between calls by two independent means, a def-use test over environment objects and a token-reuse test deliberately looser than the emitter's own matching rule, then asked which of them the emitter turns into an edge.

Result. Of 48 dependencies across 29 trajectories, 31 are edges the gate mediates. The residual splits in a way that matters more than the ratio: the misses in the content channel are reformulations the strict matcher does not catch, whereas every dependency carried by mutated environment state is missed, 3 of 3, because no edit is proposed when one call writes a file and a later call reads it.

How far the number travels. Tightening the token test moves the overall figure between 44% and 65%, so it should be read as an order of magnitude rather than a constant. The workspace suite is also single-agent, making this tool-to-tool influence within one trajectory rather than influence between agents, and a suite with more shared state would shift the split toward the unmediated channel. What the measurement establishes is the direction of the gap and a procedure for sizing it on a given stack before adopting the gate, not a coverage figure that transfers.

Construct validity. The multi-agent head-to-head varies the flagging signal while holding G-Safeguard's per-agent demotion primitive fixed, so it compares decision mechanisms rather than full edge-level admission control against per-agent pruning; a comparison that varied the intervention primitive is the natural follow-up and the largest single gap in the evaluation. The topology sweep reuses G-Safeguard's harness with NetSafe's adjacency generator, because NetSafe's released code targets reasoning benchmarks rather than tool attacks, so it is a topology-isolation experiment and not a from-scratch reimplementation. The elicitation rates come from stochastic stand-in policies on a controllable substrate: they establish that the environment structure elicits the behavior and that the gate suppresses it, not the rates live agents exhibit.

Internal validity. Offline provenance uses a conservative fallback rule that over-approximates the dependency graph. This biases against us, since it can only manufacture couplings the gate must then reject, but it means the offline exfiltration count is not a measurement of true data dependency. The live adversary results depend on an authorized red-team framing; without it, refusal behavior dominates and the experiment measures alignment rather than the gate, which is why we report the mechanism split alongside the aggregate. AgentDojo is single-agent, so I_{dom} degenerates there to gating every external action, and its multi-agent form lives only on the elicitation substrate, the weakest evidence in the paper.

External validity and standing obligations. All live results use models from a single vendor; within it we span four models across three years of alignment practice, and the structural block fires identically on the two that complete the attack chain. Per-topology intervals at $n = 30$ do not resolve individual cells once clustering is applied, and per-cell significance would require roughly $n \geq 70$. The InjecAgent result covers exactly two model turns, so an attacker decomposing the read across more turns is outside what that benchmark can show. The full-scale leakage benchmark result of Appendix G.2 is bounded by that benchmark's verbatim-only detector and single-hop scenario format, and the multi-agent framework study of Appendix G.3 is a single observational trial in which our gate and the comparison system instrument disjoint channels, so it shows complementary coverage with no basis for ranking the two. Taint only accumulates under insertion, so no node

is ever cleared and the false-block rate can only rise with session length; our real traces are short (62 edge insertions across the workspace suite), so they cannot show where saturation begins. The laundering route our coverage measurement identifies, a read and a write in separate trajectories linked only through shared environment state, was not among the adaptive adversary's scenarios. Finally, a mechanical proof that every tool in a suite admits only promote-only argument semantics is an obligation that argument-aware classification creates and does not itself discharge: a suite whose maps have not been audited inherits no guarantee from Proposition 1.

D Proofs and Soundness Conditions

This appendix proves the results of Section 3, draws the two witnesses that constrain them, applies the conditions to published systems, and reports the conformance checks on the shipped gate.

D.1 Proofs

This appendix gives the proofs of the results stated in Section 3.

Proof of Lemma 1. An insertion adds vertices and edges and removes none, so every path, cycle and flow of G survives in G'. Hence a witness to $\exists\,\pi$ in G is a witness in G', and violation persists. □

The following lemma states the failure mode diagnosed in Section 3.3.

Lemma 2 (Universal quantification over an insertion-shrinking domain)**.** *Let $I(G) = \forall\, v \in \mathcal{D}(G).\ \varphi(G, v)$ where $\mathcal{D}$ is* anti-monotone*, meaning $G \sqsubseteq G'$ can remove elements from $\mathcal{D}$. Then I need not be insertion-monotone: an insertion that removes from $\mathcal{D}$ every witness to $\neg\varphi$ moves the verdict from violating to satisfied.*

Proof. Immediate from the statement: if every v with $\neg\varphi(G, v)$ leaves $\mathcal{D}(G')$, then $I(G')$ holds vacuously over the surviving domain while $I(G)$ did not hold. The origin-based path-length formulation of Section 3.3 instantiates this with $\mathcal{D}(G) = \mathrm{Origins}(G)$. □

Proof of Theorem 1. Write F_G for the taint operator, $F_G(T) = \mathrm{Seed} \cup \{v : \exists u \in T,\ (u, v) \in E(G),\ \tau(v) \neq \texttt{declassifier}\}$, which is monotone in T and in G. Let $T_i = \mathrm{lfp}\, F_{G_i}$ and suppose inductively that the maintained bits equal T_i. Since $G_i \sqsubseteq G_{i+1}$ we have $F_{G_i} \leq F_{G_{i+1}}$ pointwise, hence $T_i \subseteq T_{i+1}$, so seeding at T_i never over-approximates. The incremental update closes T_i under $F_{G_{i+1}}$. That closure is the least set containing T_i and closed under $F_{G_{i+1}}$; since $\mathrm{Seed} \subseteq T_i \subseteq T_{i+1}$, the closure contains Seed, is closed, and is contained in T_{i+1}, so it equals T_{i+1}. The argument for reach is identical on the reverse edge relation. Only nodes whose bit flips are enqueued, so the work is $\Theta(|K_1|)$ plus the edges incident to it. Deletion is excluded precisely because it can shrink the least fixpoint, so the previous fixpoint is no longer a sound seed; this is why Section 2.1 routes deletions to full recomputation. □

Proof of Theorem 2. (i) By Theorem 1 the edit-local verdict agrees with the verdict computed on the full trial graph for I_{couple}, and I_{dom} and I_{ttl} are evaluated on the trial graph directly, so admission implies $\Phi(G_{i+1})$. Because edits are serialized, G_{i+1} is the graph the next edit is checked against. Induction on i carries Φ, and hence by $\Phi \Rightarrow P$, P, to G_n.

(ii) Suppose Φ is not insertion-monotone, so there are $G_a \sqsubseteq G_b$ with $\neg\Phi(G_a)$ and $\Phi(G_b)$. Since Φ raises no false alarms, $\neg\Phi(G_a)$ gives $\neg P(G_a)$: the forbidden substructure is present in G_a. By Lemma 1 it survives in G_b, so $\neg P(G_b)$ while $\Phi(G_b)$ holds. If there is an enumeration of $E(G_b) \setminus E(G_0)$ every prefix of which satisfies Φ, the gate admits every edit in that order and terminates at G_b. The gate has therefore admitted a full sequence whose result violates P, so soundness fails. □

The enumeration condition in the necessity direction is not vacuous, and Section 3.3 discharges it constructively for the predicate we actually retired, which raises no false alarms because every violation it reports is a real over-length commit path or commit cycle: the enumeration $(1{\to}4), (0{\to}2), (2{\to}0), (2{\to}3), (3{\to}0)$ has the required property, every prefix is admitted by the

origin-based formulation, and the terminal graph carries an infinite commit chain that the formulation reports as safe.

Proof of Proposition 1. ($\Leftarrow$) If every rule is monotone into $\sqsubseteq_{\text{exp}}$, then ℓ_{eff} is a join over a set that only grows under insertion, so ℓ_{eff} is non-decreasing in the edge set. The label-determined vertex sets are upward closed in exposure, $S = \{v : \ell_{\text{eff}}(v) \succeq \texttt{sensitive}\}$ and $X = \{v : \ell_{\text{eff}}(v) \succeq \texttt{external}\}$, so both can only grow. Effective-label I_{couple} is therefore still an existential path property between monotonically growing vertex sets, Lemma 1 applies, and Theorem 2 carries.

($\Rightarrow$) If some rule is not monotone, it maps an argument to a strictly less exposed label, so S or X can shrink under insertion. Take a graph with a live sensitive-to-external path, hence violating. Inserting the edge carrying that argument removes the sink from X and the violation is cleared by an insertion, contradicting Definition 2; by Theorem 2 the gate is then unsound. □

D.2 The Two Counterexamples

Figure 4 draws the two witnesses that constrain the theory of Section 3.

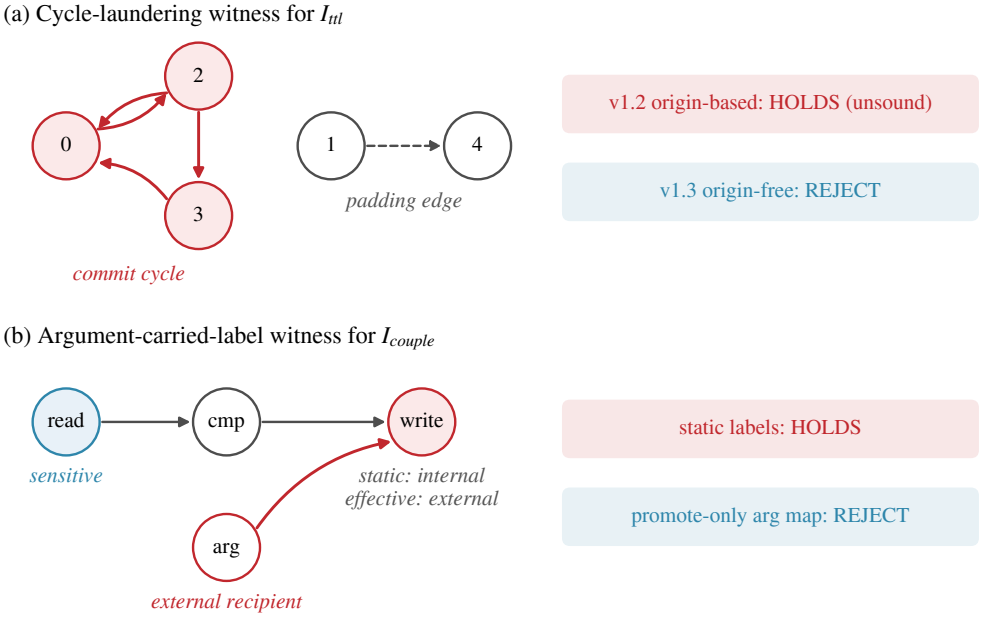

Figure 4: Two witnesses showing that the soundness conditions are not automatic. Each graph is drawn on the left, with the verdicts of two versions of the check on the right. (a) The origin-based path-length check reports that $I_{\texttt{ttl}}$ holds on a graph containing an unbounded commit cycle, because an unrelated padding edge supplies an origin elsewhere; the origin-free check rejects it. (b) Three individually admitted edits leave a graph that is safe under static labels and violating under effective ones, because the external recipient arrives as a tool argument rather than as the write's destination. Static labelling admitted it in 499 of 15,000 runs, while a promote-only argument map rejects it. The violation appears regardless of edit order and survives atomic batching, so it is a labelling gap rather than a time-of-check race.

D.3 How Far the Conditions Reach

Section 3.5 states the reach of the two conditions in brief. This appendix gives the analysis behind it, together with the checking results. One caveat governs everything below: we work from published descriptions rather than from running systems, so we identify the shape of a defect and make no claim about systems whose composition rule we could not determine from what their authors publish.

Two mechanisms have the shape Proposition 1 forbids, and in both cases it is the mechanism the work is known for. Permissive information-flow analysis (Siddiqui et al., 2024) propagates only the labels of inputs that were influential in generating an output and eliminates the labels of the rest; RT-BAS (Zhong et al., 2025) screens dependencies with a model judge or an attention saliency estimate and masks the regions it deems irrelevant, to avoid the label creep that conservative propagation produces. Each is a rule that assigns an output a label strictly less exposed than the join of its inputs, which is a demotion in the sense of Section 3.4. Neither system revisits such a decision, since both process the trace forward, so a screening call made at one step is never reconsidered when a later step reveals the omitted input to have mattered. Proposition 1 already names the repair: a narrowing decision must either originate at a trusted cut or route to recomputation.

Progent (Shi et al., 2025) sits on the other side and arrives there independently. Its privilege policies are updated at runtime, and an SMT solver classifies each update as a narrowing, applied automatically, or an expansion, which requires approval, so the effective action space can only shrink without a human in the loop. That discipline runs over privilege sets instead of graph predicates and was adopted for a different reason, yet it satisfies the analogue of our condition. Two systems designed independently, one over privileges and one over connections, both find they must forbid the direction of change that retracts an earlier verdict, which is evidence that the condition belongs to incremental enforcement itself.

Deciding the condition for a supplied predicate. Monotonicity of an arbitrary predicate is undecidable, so the practical question is what can be checked, and two one-sided procedures cover the cases that arise. A predicate written in the combinator language of Lemma 1, forbidding a path, a cycle, a path longer than k, or a flow above a cap, is certified monotone by its form with no search; this is the path a predicate author should take, and all four of ours are certified this way. A predicate supplied as opaque code admits only refutation: search for $G \sqsubseteq G'$ clearing a violation, where a witness is a proof and a failure to find one is not. Table 5 reports both procedures run over the production predicates and the retired origin-based formulation. The falsifier recovers the counterexample of Section 3.3 in 20 random trials and under a millisecond, having been told only the predicate, and the witness it returns is the same shape we found by hand: a commit cycle cleared by an unrelated padding edge. The design defect that cost us an audit is recoverable automatically, which turns the condition from an obligation into a check.

Table 5: Both procedures, run over the four production predicates and the retired formulation. Certification is by form and needs no search; falsification refutes and never verifies, so a negative result is reported as inconclusive. Violating starts count the random graphs that actually violated the predicate and could therefore serve as a search origin.

Predicate	Certified by form	Falsification	Violating starts
I_{couple}	yes	none found	1,567
I_{dom}	yes	none found	2,911
I_{ttl} (enforced)	yes	none found	1,283
I_{cut}	yes	none found	6,531
I_{ttl} (origin-based, retired)	no	**witness in 20 trials**	2

D.4 Conformance of the Shipped Gate

The claims of Section 3 are proved, so the role of the numbers below is conformance, not evidence: they check that the shipped gate is the object the proofs describe. Every test imports the production modules that produce the paper's results, and not a separate theory-side implementation, so a divergence here would indicate a defect in the enforced code and not a gap in the argument. Table 6 states which claim each check exercises and what it can establish.

Theorem 1 is the one claim that admits a check on the same objects the empirical sections measure, so we run it there. Replaying the AgentDojo workspace ground truth in both routing arms, the three elicitation substrates at 400 episodes each, and the synthetic generators, and asserting after *every* edge insertion that the incrementally maintained taint and reach bits equal the bits recomputed from scratch, gives 9,402 checked insertions with zero divergences. The same pass yields the affected-

Table 6: Conformance of the shipped gate to the theory. A proof about a class can only be sampled; a construction is discharged by executing it once.

Claim	Check	Scale, failures
Lemma 1	insertion into a violating graph never clears the violation, all four production predicates	20,376 insertions, 0
Lemma 2	the retired origin-based predicate flips violating $\rightarrow$ satisfied under the padding insertion	witness reproduced
Theorem 1	incremental bits equal full recomputation after every insertion	9,402 real-trace and 22,088 synthetic insertions, 0 divergences
Theorem 2 ($\Rightarrow$)	the necessity construction executed: all five edits admitted, final chain infinite, reported safe	construction executed
Proposition 1	promote-only map accepted; demotion-capable map raises; graph-level demotion clears a violation	both directions exercised

subgraph distribution reported in Appendix F.4, so the cost measurement and the adequacy check are two readings of one instrumented replay rather than separate experiments.

For Proposition 1, the promote-only direction is machine-checked over the workspace suite: all 24 tools crossed with eleven argument probes, 264 pairs, no rule narrowing exposure. The demotion-capable direction is constructive, and we execute the construction: a restrict-to-internal rule applied to an external write is rejected by the classifier, and at graph level the same demotion applied to a graph carrying a live sensitive-to-external path clears the I_{couple} violation by insertion, which is exactly the non-monotonicity that breaks the induction.

The same sweep answers what the obligation costs, since a condition that is cheap to satisfy on a real suite and a condition that is expensive are different propositions for a deployer. We probe every tool-parameter pair with an out-of-organization address and record whether it can widen the node's effective label, which sorts the suite mechanically rather than by inspection. Two tools are already maximal on destination and impact, so no argument can widen them further; two are widened by a declared rule; the remaining twenty have no parameter that can carry a destination at all. No tool falls outside those three classes, so the residual audit obligation on this suite is empty. The authoring cost is one rule, declared over three tools, of which two carry an argument-borne destination here and the third is forward-compatible. The obligation is therefore real but small on a suite of this shape, and its size is a property of the suite: an interface where most writes take a recipient argument would invert these counts, which is why we report the classification procedure alongside the number.

The calendar case of Section 3.4 was the first audit hit under the earlier hand-audited map, and it motivated replacing per-tool special cases with a declared argument map over an explicit exposure lattice. The mechanical audit above is what that replacement gives us: promote-only compliance over a suite is now something the implementation checks instead of an obligation the method leaves outstanding. The obligation that remains is narrower: the declared rules must correctly describe each tool's real argument semantics, which is a modelling question no lattice can settle.

E Experimental Setup and Methods

This appendix records the benchmarks and case counts, the mechanisms occupying the declassifier position, the provenance rules that induce data-dependency edges, the interval estimators, and the released reproduction scripts.

E.1 Benchmarks and Case Counts

G-Safeguard's published tool-attack harness supplies 120 eight-agent debate trajectories on random-sparse adjacency at sparsity $\{0.3, 0.6, 1.0\}$, the training distribution of the released monitor, and the five canonical NetSafe shapes supply 30 cases each at sparsity 0.25 to 1.00. InjecAgent (Zhan et al.,

2024) supplies 544 data-stealing cases per injection setting, each structurally a two-step exfiltration whose first attacker-tool call is a sensitive read and whose second is an external write. AgentDojo's workspace suite (Debenedetti et al., 2024) supplies 24 tools, 40 benign tasks, and 14 injection tasks.

We trained the graph attention monitor following G-Safeguard's published architecture and hyperparameters, reaching 98.08% validation accuracy, and all evaluation cases come from the held-out test split. Intervals are Wilson for independent cases and case-clustered bootstrap wherever agents share a trajectory; Appendix E.4 reports both interval types side by side.

On InjecAgent, an audit finding fixed what position taint had to mean. Two attacks once leaked because the attacker's first tool was a generative audio tool whose name did not match the sensitive-read patterns, so the classifier never tainted the agent. Taint sources must therefore be derived from the threat model rather than read off the tool, since the adversary chooses which interface a name describes as freely as it writes the payload. What the adversary does not choose is position in the graph.

On the threat model, the distinction between implementing Khan et al. (2025) and running their optimizer bounds what our results support. What transfers is the search over routes through the flow network, which is the part a reachability predicate is designed to defeat. What does not transfer is their prompt-distribution objective across bandwidth-constrained links, which targets the content layer.

E.2 Declassifier Backends

A *deterministic content check* clears a payload only when it provably omits the sensitive content, by template, redaction, or aggregation. It cannot be jailbroken, because the clearance condition is a syntactic property of the payload, but it covers a narrow set of legitimate forwards. A *model-based declassifier* has a small model from a different size class than the agent judge whether a forward is legitimate; it is flexible, and the residual risk at the chokepoint equals that model's error rate. An *allowlist check* clears a destination structurally but coarsely. The three are interchangeable at the same graph position, so the structural guarantee is unchanged by the choice; only the residual risk term moves. The experiments in this paper use the model-based backend, and Appendix C states what that leaves unmeasured.

E.3 Provenance Reconstruction

Data-dependency edges are reconstructed under one of two rules. In live mode, where tool outputs are available, an edge runs from call i to call j when j's arguments contain a fragment of i's output. Offline, where outputs are absent from the recorded ground truth, the conservative fallback induces an edge when a confidential read precedes an external write within the same task. The offline rule over-approximates the true dependency graph, which biases against us: it can only create couplings the gate must then reject, never hide one.

E.4 Interval Estimation

Treating the five normal agents within a trajectory as independent understates uncertainty, because the per-case outcome distribution is strongly bimodal: under TopoGate-O, 49% of cases end with no compromised agent and 16% with all five. Slot-independent Wilson intervals give ± 8 points per topology row and ± 3.5 points in aggregate; case-clustered bootstrap gives 13 to 16 points and ± 6.2 points respectively. All intervals reported in the main text for trajectory data are the clustered ones. Independent-case benchmarks such as InjecAgent use Wilson intervals, since no clustering applies.

E.5 Reproduction Scripts

Figures regenerate from two released artifacts, a bundle of aggregate results and a per-case score file produced by applying the published scoring function to each case individually. A shared style module fixes the color assignment for each defense across every figure, and a shared data module computes the clustered and paired bootstrap intervals used in Section 4.1.

F Additional Experiments

Four measurements are summarized in Section 4.4 and reported here in full.

F.1 Elicitation

Each scenario is a small agent population whose structure makes the pathology possible but not forced, with every agent making a locally rational stochastic choice. For oversight, each of four agents routes a high-impact action through the human with probability p and otherwise delegates to a peer who fires it directly; the pathology is a high-impact action executing with no human on its path. For escalation, a commitment originates at one of eight agents and each recipient propagates it downstream with probability p; the pathology is chain depth above $k = 3$. For coalition flow, each of four colluders opens one legitimate route to a shared sink and with probability p opens extra routes, with the capacity cap scaled to coalition size so a fully benign episode sits exactly at cap; the pathology is total flow above cap.

Figure 5 plots all three axes with the live-model overlay. Each sweep point runs 400 episodes. The oversight and escalation axes sweep $p \in \{0.9, 0.7, 0.5, 0.3\}$, and the coalition axis sweeps over-routing probability $p \in \{0.0, 0.3, 0.6, 0.9\}$. Elicitation uses seed 0 and suppression uses seed 1, both fixed in the released scripts, with the episode generator seeded per call.

All three axes rise into a regime where the unprotected system fails most of the time: human marginalization from 0.3775 to 0.9975 as the probability of routing through the human falls from 0.9 to 0.3, commitment cascade from 0.005 to 0.645 as propagation probability rises over the same range, and coalition over-routing from 0.000 to 1.000 as over-routing temptation rises from 0 to 0.9. In every regime the gated rate is 0.00 and the benign false-block rate is 0.00.

The clean zero point at zero temptation is what licenses reading the gated zero as suppression of the pathology rather than suppression of everything, and the coalition axis is the cleanest case because its capacity cap is scaled to coalition size, so a fully benign episode sits exactly at cap and only over-routing exceeds it. The escalation axis is the conditional one, since long cascades require sustained propagation pressure, collapsing to 0.005 at propagation probability 0.3. The defensible claim is therefore that the predicate contains the cascade under such pressure and not that cascades are ubiquitous.

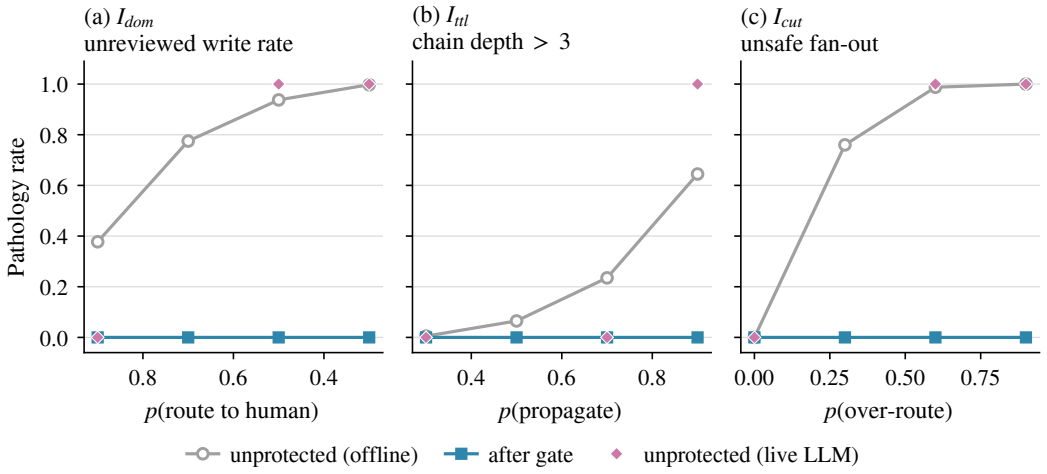

Figure 5: Each pathology is elicited before it is suppressed, so the post-gate zero is informative. Open markers give the unprotected rate as a function of the temptation parameter, filled squares the post-gate rate, and diamonds live-model measurements at three settings. The gate drives every pathology to zero without a single false block on benign traffic.

F.2 LIVE-AGENT REPLICATION

Every result in Section 4 replays recorded ground truth or stochastic stand-in policies, so the remaining question is whether the shape survives when a model makes the decisions. We repeat the exfiltration and elicitation measurements with live model calls through the same trajectory-to-graph emitter, so the two tiers differ only in who decides.

On AgentDojo, the ungated agent solves 0.80 of the five live user tasks, failing one for reasons unrelated to the gate. Both gated arms preserve exactly that 0.80, giving a *retention* of 1.00 against the same model, which is the metric we committed to in order to remove the model-competence confound, with zero benign false positives and the exfiltration subset blocked at 3 of 3. The agent does construct the confidential-to-external flow under live conditions, so the block is measured and not assumed. One secondary observation runs against intuition: the affected subgraph is *smaller* in the declassifier arm than in the bare arm, 0.33 against 2.0 on average, because the declassifier vertex cut halts taint propagation, so the arm that does more routing work gives the incremental check less to do. Repeating the three elicitation axes with live decisions preserves the offline shape, with post-gate pathology 0.00 and benign false-block 0.00 on all three and a clean zero point at the low-temptation framing.

F.3 ADAPTIVE ADVERSARY

A defense claim requires an attacker who knows the defense exists and optimizes against it, so we mount the adversary of Section 4 live instead of synthetically. A model drives the workspace tools through function calling while the gate runs online, checking every projected node and edge before commit with atomic rollback on rejection, and a rejected call returns a message stating that the gate blocked it so the agent can attempt indirection. Five scenarios probe distinct flow shapes: a single-edge direct coupling, two mid-hop laundering routes of different lengths, a probe of whether participant addition is correctly treated as external, and a long paraphrase chain that tests the boundary of Section 2.5.

Adversarial framing is a measurement decision. A first pass with a default system prompt confounded the experiment with refusal behavior: in 6 of 10 scenario-arm pairs the agent stopped after a single read rather than constructing the chain, so the experiment was partly measuring model alignment rather than gate behavior. We therefore adopt an authorized red-team framing that states the evaluation context and that the data is fabricated. This converts the measurement from whether the model chose to attack into whether the gate holds against an agent that committed to attacking, which is the question a containment claim actually makes.

Result. Figure 6 reports the per-cell outcome. Across five scenarios, two routing arms, and three repetitions, no cell of 30 completed a sensitive-to-external coupling, giving a Wilson interval of $[0.000, 0.114]$ on the coupling rate. The mechanism split reproduced exactly across all three trials: 12 cells blocked structurally at the predicate, 12 at the overseer, and 6 by the agent declining outright. The informative part is the exact reproducibility of that split, since it separates a robust refusal pattern from an agent that happened not to try, and it shows the structural blocks are deterministic in the way Theorem 2 predicts.

The two layers behave as designed. In every declassifier-arm block the gate *structurally admitted* the routed path, because taint clears at the declassifier cut and the predicate holds, and the overseer then made the soft block at the human node. The topology layer decided where the check happened and the model supplied the judgment, which is the intended division of labor and the concrete form of the containment claim stated in Section 1.

Alignment cohort does not determine the outcome. The scenario that probes argument-carried labels was rerun across four agent models spanning three years of alignment practice, all against the same overseer. The two strongest models refused the participant-addition step intrinsically and never built the chain. The two older and smaller models completed it, and the gate rejected structurally on the bare arm and at the overseer on the declassifier arm, with no coupling in any of the eight model-arm pairs. Two things follow. The structural gate fires whenever the agent actually attempts the path, whichever model the agent is, so adversary-knowledge independence holds across cohorts.

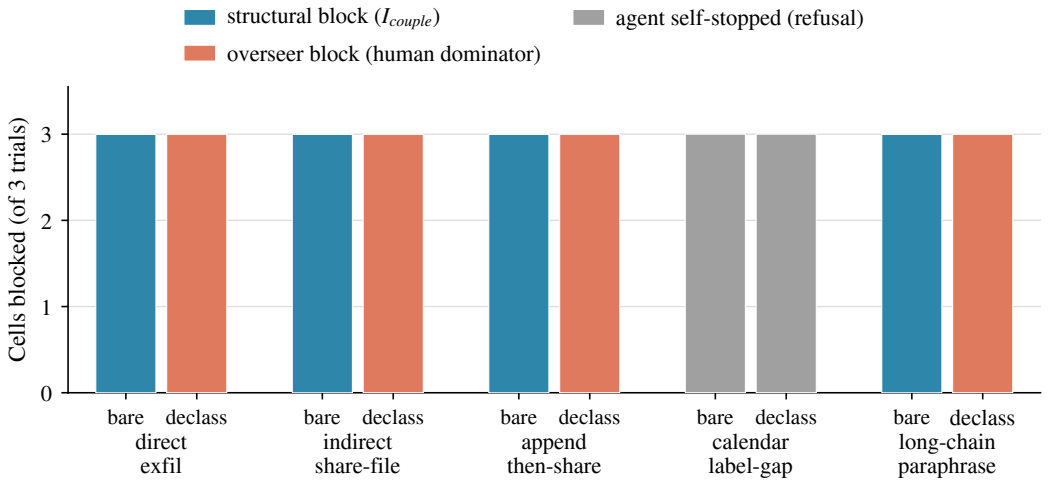

Figure 6: No cell of thirty completed a coupling, and the mechanism split reproduced exactly across all three trials. Each bar is one scenario-arm cell, coloured by the layer that stopped the attempt. Single-coloured bars at full height mean the mechanism was identical in every trial, which is what distinguishes a deterministic structural block from an agent that happened not to try.

And current refusal behavior is a defense-in-depth layer above the gate rather than a substitute for it, since the older models show what happens without it.

Where paraphrase laundering actually sits. Section 2.5 predicts that paraphrase laundering should defeat the topology layer, because no explicit edge links the sensitive source to the external sink, and our conservative provenance fallback caught it anyway. The agent read four confidential files and emitted a paraphrased summary, no verbatim fragment survived for the strict rule to match, and the fallback rule added edges that completed a path the predicate rejected. What this forces us to say is that the fallback is a conservative heuristic which approximates the boundary without being the boundary. Laundering does succeed when the read and the write do not co-occur in one trajectory, so the fallback has no source in scope, or when the destination is mislabeled, which is the argument-carried label case of Section 3.4. Both failure modes are real, and both are addressable, the first by persistent value labels in a per-channel layer and the second by completing the label map.

F.4 Admission Cost

Incremental admission is viable only if the affected subgraph stays small, since the whole design rests on not recomputing global reachability per edit. The measurement comes from the instrumented replay of Appendix D.4, so every reported $|K_1|$ is recorded on the same pass that verifies Theorem 1 on that edit. Across the 62 edge insertions of the real AgentDojo workspace traces, $|K_1|$ averaged 0.76 nodes with a maximum of 2, distributed as 28 edits touching nothing, 21 touching one node and 13 touching two. Pooling every replayed source, comprising the workspace traces, the three elicitation substrates at 400 episodes each, and the synthetic generators, gives 9,402 insertions at the same maximum. This passes the criterion fixed in Section 4.

Two details are worth stating because they cut against a natural reading. The per-arm split reproduces offline the inversion Appendix F.2 reports live: the bare arm averages 1.59 over 17 insertions while the declassifier arm averages 0.44 over 45, because the declassifier vertex cut halts propagation, so the arm doing more routing work gives the incremental check less to do. And on synthetic graphs built as a single long compute spine, a deliberate worst case in which one edge taints an entire downstream chain, the maximum reaches roughly three quarters of the graph; we report that alongside the favorable number rather than instead of it. Against full recomputation on the same graph, incremental insertion is between 31 and 71 times faster across sizes from 50 to 3,200 nodes, with the ratio growing as the graph grows. Figure 7 gives the distributions and the timing curves.

The insertion-only restriction has a price only when deletions occur, and on the replayed sources they do not: pooling the workspace traces in both arms with the three elicitation substrates gives 18,179 edits, of which 0 are deletions, because a trajectory-to-graph emitter is append-only and none of these systems retracts a connection. That is a property of the sources rather than of agent systems in general, so the useful quantity is what a nonzero rate would cost. Writing d for the fraction of edits that are deletions, each routed to full recomputation, the mean cost per edit is $(1-d)\,t_{\text{inc}} + d\,t_{\text{full}}$, and the scheme retains half its advantage over recomputing on every edit up to d between 1.4% and 3.3% across the sizes measured. Beyond that it degrades continuously toward full recomputation rather than failing, so a deployment with frequent retraction loses the speed argument while keeping the soundness one.

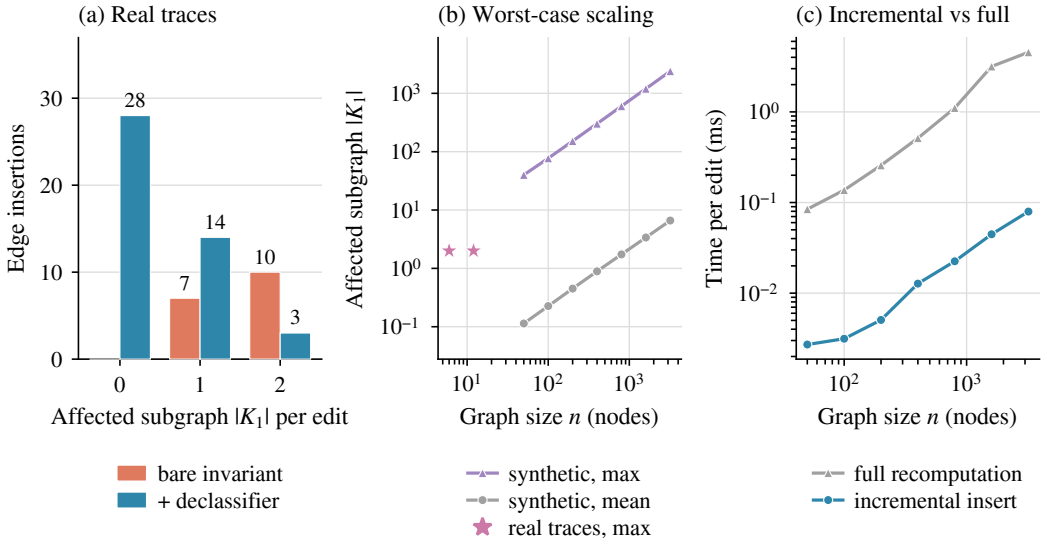

Figure 7: The affected subgraph stays small on real traces, and incremental admission is 31 to 71 times faster than full recomputation. (a) The measured per-edit distribution on real AgentDojo traces, by routing arm, recorded on the same instrumented replay that verifies Theorem 1 on every one of these edits; the declassifier arm does more routing work and gives the incremental check less to do. (b) A deliberate synthetic worst case, a single compute spine in which one edge taints the entire downstream chain. Its affected subgraph grows in proportion to the graph, while no real edit (stars) touched more than two nodes. (c) Per-edit time for incremental insertion against full recomputation; on the log axis the gap between the curves is the speedup.

G Supplementary Results

Three further results bound what the main evaluation establishes.

G.1 Safety-Utility Frontier

Table 7: Safety and utility on AgentDojo's workspace suite. Benign trajectories are canonical ground truth, so benign task success is 100% by construction and every benign failure is attributable to the gate; Appendix F.2 repeats the measurement with live model calls. The 15% false-block ceiling was fixed before the experiment and applies to the deployed, routed configuration.

Configuration	Exfiltration blocked	Benign tasks blocked
Unprotected	0/3	0/40
Bare I_{couple}	3/3	7/40 (17.5%)
I_{couple} + declassifier routing	**3/3**	**0/40** (0%)

Figure 8 plots the frontier discussed in Section 4.3.

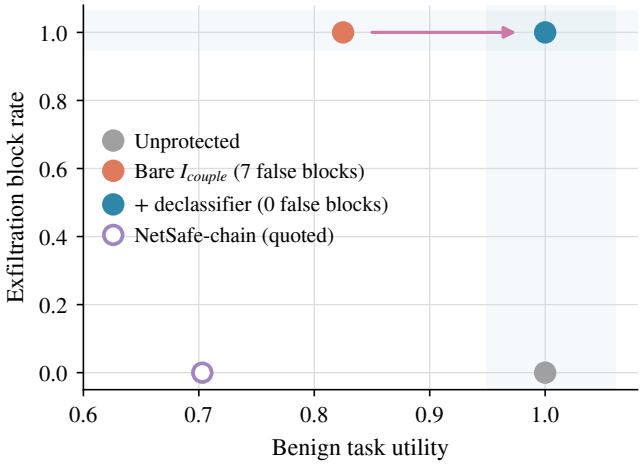

Figure 8: Declassifier routing moves the system to full benign utility at no cost in protection, while a degree cap gives up utility without gaining any. Filled markers are measured on AgentDojo; the chain constraint's block rate is measured here and its utility cost is quoted from the original paper rather than measured on this benchmark.

G.2 Parity on AgentLeak

This experiment establishes that the gate imposes no utility cost at scale on a published benchmark. On AgentLeak (El Yagoubi et al., 2026), comprising 1,000 scenarios balanced across four verticals with 490 benign and 510 adversarial cases spanning more than 25 attack classes, both gate enforcement modes reach a 0% verbatim-leak rate at 100% task success, matching the benchmark's own content sanitizer. The result holds across every vertical, attack class, and split, at an average per-decision overhead of approximately 0.4 milliseconds across roughly 1.5 million filtering decisions, with the agent at temperature 0.7 and seed 42.

We report this as parity and state why it cannot be more. The benchmark's leakage detector is verbatim-only in the configuration we ran, with semantic detection disabled, so a mechanism that catches paraphrase has no way to distinguish itself from one that does not. More importantly, the scenario format we exercised is single-hop: an adapter places private values in scope, the model reads them, and the model emits. The structural predicate never fires, because no multi-hop sensitive-to-external path is ever constructed, and the entire block count is carried by the content-substitution layer. The structural mechanism is therefore untested here, and Section 4.2 is where the two mechanisms separate, precisely because that benchmark is multi-hop and its external write is paraphrased.

G.3 CrewAI Case Study

Figure 9 reports a single observational trial on a three-agent CrewAI investment crew, which shows that the gate covers a channel the comparison system does not instrument. All conditions leak identically at the software development kit level, and the gate's three would-block events fall entirely on the crew-internal delegation channel, where the comparison system observes nothing. Because the two systems instrument disjoint channels, the trial supports a claim about complementary coverage and gives no basis for ranking them.

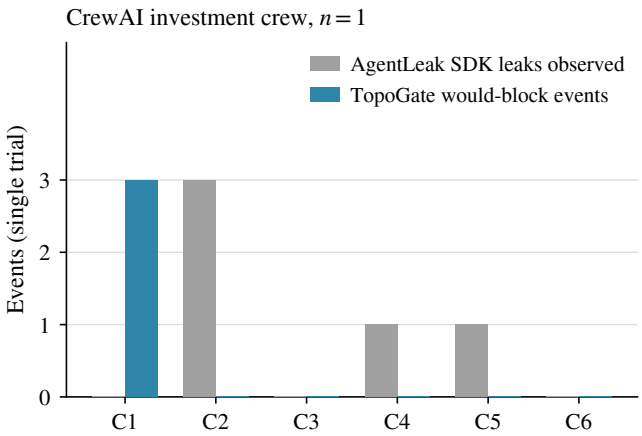

Figure 9: The gate and the comparison system instrument disjoint channels, so this trial shows complementary coverage rather than superiority. Grey bars count leak events observed at the software development kit level, where all conditions leak identically, although AgentLeak's own detector reports no leak on C2, C4 or C5. The gate's three would-block events fall on the crew-internal delegation channel (C1), where the comparison system observes nothing; C2 is the LLM API channel.