GUARDCIRCUITS: LOCAL COMPOSITION AND GLOBAL PLACEMENT FOR AGENTIC PRIVACY

Anonymous authors
Paper under double-blind review

ABSTRACT

As multi-agent systems take on real workflows spanning agents, tools, memories, and external services, protecting sensitive information becomes increasingly important while preserving the information needed to complete the task. Many guard mechanisms have been introduced to preserve privacy. Existing work has largely focused on improving individual guards, while placement is treated locally, for example at a particular tool call or system boundary, without considering the global graph structure and the joint tradeoff between privacy, utility, latency, and cost in their design. We therefore ask a more architectural question: *what behaviors and design principles emerge when imperfect guards are composed, and where should the resulting compositions be placed in an agent workflow?* We introduce *GuardCircuits*, a framework inspired by electrical circuits, that composes guards using series and parallel structures together with deterministic or stochastic switches. We show that a guard circuit admits an equivalent privacy–utility–latency–cost signature and, in a high-reliability regime, a tropical (min–plus) resistance law that reveals depth–width tradeoffs between privacy and utility. We then study global placement: ideal mediation reduces to a directed cut problem, while imperfect guards can be arranged in ordered barriers whose privacy resistance accumulates across successive lines of defense. Empirical results on privacy benchmarks support these composition laws and show that minimum-cut placement reduces leakage, giving a principled basis for both arranging and placing guards in agentic systems.

1 INTRODUCTION

As multi-agent systems take on real workflows that span agents, tools, memories, and external services, they must protect sensitive information while preserving the information needed to complete the task. Leaking such information can violate regulations such as GDPR (Voigt and Von dem Bussche, 2017) and HIPAA (HHS, 1996), yet language model (LM) agents have been shown to leak private information in many settings (He et al., 2025; Shao et al., 2024; Goel and Gurevych, 2026).

Guards that filter or censor messages along sensitive edges can reduce this risk, but they are imperfect (Staab et al., 2025; Jeon et al., 2026; Dou et al., 2024; Abdelnabi et al., 2025). A guard can miss sensitive information, and it can also remove information that the task needs (Park et al., 2026; Li et al., 2025; Bassani and Sanchez, 2025). Composing guards shifts this tradeoff: for example, passing a summary through a second guard before it reaches an external tool offers another chance to remove a secret, but also another chance to lose necessary context.

Where guards are placed matters as much as how they are composed. In multi-agent systems, sensitive information can travel through internal messages, tool arguments, and shared memories before reaching an unauthorized agent, so leakage depends on the communication topology (Liu et al., 2026; Yagoubi et al., 2026). A guard cannot protect a route that bypasses it, guarding every edge is costly, and repeated interactions give an imperfect guard repeated chances to fail. Yet existing work largely improves individual guards and treats placement locally, such as at individual tool calls or system boundaries (An et al., 2025; Abdelnabi et al., 2025).

These observations lead to two questions:

Q1. How should guards be combined?	**Q2. Where should guards be placed?**
Trade leakage against utility, latency, and cost.	Cover every leak route without guarding every edge.

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.

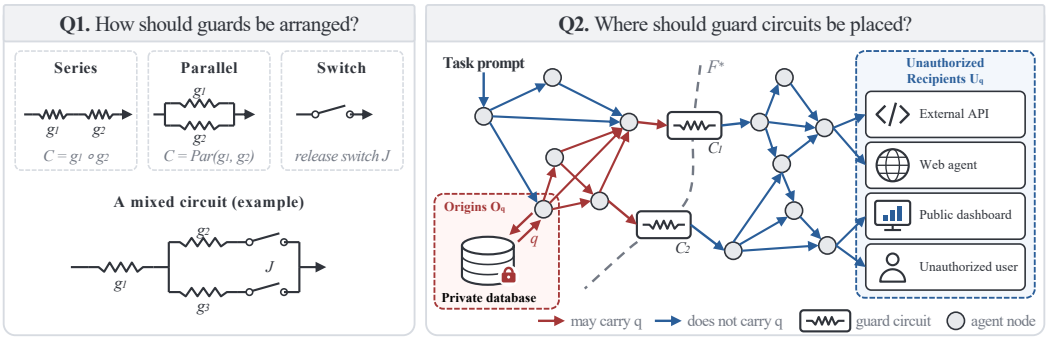

Figure 1: **Overview of GuardCircuits. Q1 (left):** guards g_i, drawn as resistors, are composed in series ($g_1 \circ g_2$), in parallel ($\mathrm{Par}(g_1, g_2)$), and with release switches J into a guard circuit C. **Q2 (right):** guard circuits C_1, C_2 on the minimum cut $F^\star$ intercept every route from the origins O_q of protected information q to the unauthorized recipients U_q.

In this work, we propose *GuardCircuits*, a framework inspired by electrical circuits (Fig. 1). We treat each guard as a resistor-like filter and compose guards into local modules called *guard circuits*. Each guard circuit mediates one local information flow, which lets us study how circuits behave separately from where they are placed in the workflow.

For Q1, we show that every guard circuit reduces to a single equivalent signature of privacy, utility, latency, and cost. Guards can be composed in series, in parallel, and with switches that control which parallel outputs are released (Fig. 1, left). For highly reliable guards, the signature follows a tropical (min–plus) resistance law: depth buys privacy, while width buys task reliability.

For Q2, we show where to place guard circuits so that every forbidden route is protected. With perfect guards, it suffices to guard a minimum cut between the sources of sensitive information and the unauthorized recipients. With imperfect guards, we unroll each execution of a possibly cyclic workflow into a causal message flow, and show that protection accumulates across the successive lines of defense along this flow.

Experiments on three agent privacy benchmarks support these predictions. Adding guards in series lowers both leakage and task success, while adding them in parallel raises both, and the leakage effect holds across six guard models. Guarding the minimum cut also reduces leakage compared to guarding the same number of random edges. In summary, we have the following **main contributions:**

- **Guard circuits.** We derive exact composition laws (Thm. 4) and a high-reliability tropical min–plus law that reveals the depth–width privacy–utility tradeoff (Thm. 13 and Cor. 5).
- **Circuit synthesis.** We prove monotone substitution (Thm. 6) and derive an exact Pareto dynamic program over budgeted guard-circuit topologies.
- **Placement.** We show that ideal guard placement reduces to a directed minimum cut (Thm. 18), while stochastic protection accumulates across ordered execution barriers (Thm. 7).
- **Experiments.** Across three agent-privacy benchmarks and six guard models, results support the predicted composition trends and show that minimum-cut placement reduces leakage relative to random placement.

2 Related Work

LM guards. Privacy guards for LMs either prompt a model to rewrite a message without its sensitive content (Staab et al., 2025; Jeon et al., 2026) or detect sensitive spans and apply a constrained edit (Dou et al., 2024; Abdelnabi et al., 2025). Recent work improves individual guards through robustness, reduced over-defense, cascades, and routing (Bassani and Sanchez, 2025; Li et al., 2025; Nagireddy et al., 2025; Lee et al., 2025). When guards are stacked, repeated redaction trades utility for privacy (Jeon et al., 2026), staged attacks can defeat safeguard pipelines (McKenzie et al., 2026), and errors are correlated across models (Kim et al., 2025). In contrast, we derive exact composition laws for guards arranged in series, in parallel, and with release switches.

Privacy in multi-agent systems. LMs and LM agents often violate contextual integrity (Mireshghallah et al., 2024; Shao et al., 2024; Goel and Gurevych, 2026), and in multi-agent systems, leakage

depends on the communication topology and occurs through internal channels (Liu et al., 2026; Yagoubi et al., 2026; Park et al., 2026). System-level defenses enforce information-flow or privilege policies on agents (Debenedetti et al., 2025; Costa et al., 2025; Bagdasarian et al., 2024), and topology-aware defenses detect and isolate compromised agents (Wang et al., 2025; Yu et al., 2025). Most closely related, MESA (Li et al., 2026) ranks communication edges for monitoring under a budget. We instead ask which edge sets separate private sources from unauthorized recipients, and how imperfect guards on those sets combine.

Information-flow control. Graph cuts have been used to place security mediators and sanitizers in programs (King et al., 2010; Livshits and Chong, 2013), and we recover minimum-cut placement as the special case of perfect guards. Reliability theory analyzes series and parallel compositions of unreliable components (Moore and Shannon, 1956; Birnbaum, 1969; Satyanarayana and Wood, 1985), and exact reliability on general graphs is intractable (Provan, 1986). We adapt this analysis to guards that must remove harmful information while preserving useful information.

3 Problem Setup and Design Puzzles

We use $G = (\mathcal{V}, E)$ for the possibly cyclic directed workflow, where $\mathcal{V}$ contains the agents, tools, memories, and other components of the system, and $(u, v) \in E$ means that component u may send information to component v. For example, a node may be a planning agent, private database, or external API, while an edge may represent an inter-agent message, tool call, or memory access. Let $\mathcal{Z}$ be the space of messages transmitted through the workflow and processed by guards.

Let $\mathcal{Q}$ be the finite set of protected information classes, such as medical or financial information. For each $q \in \mathcal{Q}$, the policy label $\chi_q(v) \in \{+1, 0, -1\}$ marks a node as an authorized source or holder, policy-neutral, or an unauthorized recipient of q. Define

$$O_q = \{v \in \mathcal{V} : \chi_q(v) = +1\}, \qquad U_q = \{v \in \mathcal{V} : \chi_q(v) = -1\},$$

and let $\mathsf{Rec}_q(z) = 1$ mean that protected information q is recoverable from message $z \in \mathcal{Z}$. Let $\mathcal{D}$ be the finite set of task demands. A demand $d \in \mathcal{D}$ specifies a source s_d, target t_d, and usability test $\mathsf{Use}_d : \mathcal{Z} \to \{0, 1\}$, where $\mathsf{Use}_d(z) = 1$ means that z contains enough information to satisfy the task, not necessarily verbatim. We apply the same test to intermediate circuit messages. In the main text we fix one protected class q and one task demand d; the full multi-class specification is in App. A.2.

Three design puzzles. Even simple guard architectures can create non-obvious tradeoffs. Figure 2 poses three representative design questions suggested by the constructions in Fig. 1.

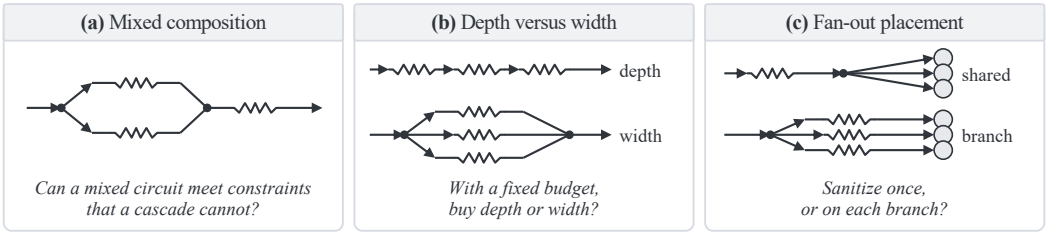

Figure 2: Three motivating puzzles. We answer (a) in Sec. 4.2, (b) in Sec. 4.3, and (c) in Sec. 5.1.

Puzzles (a) and (b) concern how to arrange guards and are answered by the circuit calculus in Sec. 4; puzzle (c) concerns where to place them and is answered in Sec. 5. Full calculations are given in App. B.1. The puzzles make clear what the formalism must accomplish; it should let us analytically understand how privacy, utility, latency, and cost depend on the arrangement and placement of guards.

4 How Should We Arrange Guards? (Q1)

We first investigate how combining guards affects their performance as a group, which we refer to as a *guard circuit*, due to its motivation through electrical circuits. A guard circuit mediates one local information flow, so throughout this section we work with the fixed class q and demand d of Sec. 3. In particular, we build a guard circuit from three elementary operations: serial composition, parallel composition, and deterministic or stochastic switches (Fig. 1, left). We also can recursively combine these primitives into richer mixed circuits, (e.g. Fig. 7). In this section we formalize this framework and study how composition affects the resulting circuit.

4.1 COMPONENTS, WIRING, AND THE TRADEOFF

Let $\mathcal{G}$ be a finite, model-agnostic library of primitive guard implementations, including deterministic rules, learned guards, human review, and hybrids. We first associate each guard with an operational signature. The choice of signature can be adapted or extended depending on the task at hand. In this work, we focus on four quantities central to our setting: harmful-information survival, useful-information survival, latency, and cost, and study how these quantities interact under guard composition. We define the primitive guard signature as follows.

Definition 1 (Primitive guard signature)**.** For a guard g, let $Z_g^{\mathrm{in}}, Z_g^{\mathrm{out}} \in \mathcal{Z}$ denote its input and output. Its operational signature is $\sigma(g) = (a_g, u_g, \tau_g, \kappa_g) \in [0,1]^2 \times \mathbb{R}^2_{\geq 0}$. Here $a_g := \mathbb{P}(\mathsf{Rec}_q(Z_g^{\mathrm{out}}) = 1 \mid \mathsf{Rec}_q(Z_g^{\mathrm{in}}) = 1)$ is harmful-information survival, and $u_g := \mathbb{P}(\mathsf{Use}_d(Z_g^{\mathrm{out}}) = 1 \mid \mathsf{Use}_d(Z_g^{\mathrm{in}}) = 1)$ is useful-information survival. The terms τ_g and κ_g denote latency and cost.

We now make the three circuit operations precise. Serial and parallel composition describe how guards are connected. A switch is a multi-input selector that receives a collection of candidate messages and decides which of them to release, possibly deterministically or stochastically. In our circuit grammar, switches are used to close parallel blocks. We first define the switch, and then combine these operations in the formal definition of a guard circuit.

Definition 2 (Switch)**.** A *switch* J is a multi-input release module. Given an ordered tuple of candidate messages $Z = (Z_1, \ldots, Z_m) \in \mathcal{Z}^m$, it chooses an index set $I \subseteq [m]$, either deterministically or according to a distribution that may depend on Z, and releases the ordered subtuple $Z_I = (Z_i)_{i \in I}$.

The choice $I = \varnothing$ denotes abstention. The switch does not rewrite the messages it releases. Examples include rule-based, score- or rank-based, uniformly random, and learned selection policies.

Definition 3 (Typed guard circuit)**.** We define typed two-terminal guard circuits recursively:

(i) Every primitive guard g is a typed guard circuit.
(ii) If the output type of C_1 matches the input type of C_2, their serial composition $C_1 \circ C_2$ is a typed guard circuit: we pass the output of C_1 to C_2, in that order.
(iii) Given circuits $C_1, \ldots, C_m$ with a common input type, $m \geq 2$, the parallel wiring $\mathrm{Par}(C_1, \ldots, C_m)$ sends the same input to all branches and bundles their outputs as the ordered tuple $Z = (Z_1, \ldots, Z_m)$. For any type-compatible switch J from Def. 2, the closed block $\mathrm{Par}(C_1, \ldots, C_m) \circ J$ is a typed guard circuit. When J is summary-compatible with privacy and utility response tables (ψ, ϕ) as defined below, we denote this circuit by $\mathrm{FJ}_{\psi,\phi}(C_1, \ldots, C_m)$, suppressing the fixed switch from the notation.

We obtain every typed guard circuit in the scalar calculus by finitely many applications of these rules.

4.2 CIRCUIT LAW

To study guard circuits without evaluating every complete architecture from scratch, we want to understand how the signature of a composed circuit can be determined from the signatures of its components. Serial composition will admit a direct rule. For a parallel composition, however, the result also depends on how the switch releases the candidate messages. We therefore summarize the behavior of a switch through two response tables.

Let $H_i = \mathsf{Rec}_q(Z_i)$ and $Y_i = \mathsf{Use}_d(Z_i)$ denote whether branch i contains protected or useful information. We call a switch J *summary-compatible* when its release behavior, as measured by Rec_q and Use_d, depends on the candidate tuple only through these Boolean patterns:

$$\psi(h) = \mathbb{P}\big(\mathsf{Rec}_q(Z^{\mathrm{out}}) = 1 \mid H = h\big), \qquad \phi(y) = \mathbb{P}\big(\mathsf{Use}_d(Z^{\mathrm{out}}) = 1 \mid Y = y\big). \tag{1}$$

Here $\psi, \phi : \{0,1\}^m \to [0,1]$ are coordinatewise nondecreasing and satisfy $\psi(\mathbf{0}) = \phi(\mathbf{0}) = 0$. Two examples make these tables concrete. If the switch releases all candidates, then under exposed-any tests $\psi(h) = \bigvee_i h_i$ and $\phi(y) = \bigvee_i y_i$. If it instead selects one candidate uniformly at random, then $\psi(h) = m^{-1} \sum_i h_i$ and $\phi(y) = m^{-1} \sum_i y_i$. The two tables describe the privacy and utility behavior of the same switch and are not, in general, independently adjustable. For the common binary exposed parallel-any block, we write $C_1 \parallel C_2 := \mathrm{FJ}_{\vee,\vee}(C_1, C_2)$, retain $\mathrm{FJ}_{\psi,\phi}$ when the switch law must be explicit, and abbreviate exposed-any privacy by $\mathrm{FJ}_{\phi} := \mathrm{FJ}_{\vee,\phi}$. Additional examples and cases requiring richer state or closed-block calibration are given in App. A.4. Serial composition accumulates latency and cost, while parallel branches execute concurrently, so latency is set by the

slowest branch and cost remains additive. Under the assumptions in App. A.3, privacy and utility admit analogous composition rules. Writing $\sigma(C) = (A_C, U_C, T_C, K_C)$, we obtain:

Theorem 4 (Equivalent signature). *Under Asm. 9, let $\sigma(C_i) = (A_i, U_i, T_i, K_i)$. Then, for serial composition,*

$$\sigma(C_1 \circ C_2) = (A_1 A_2,\ U_1 U_2,\ T_1 + T_2,\ K_1 + K_2), \tag{2}$$

and for $C = \mathrm{FJ}_{\psi,\phi}(C_1, \ldots, C_m)$,

$$A_C = \sum_{h \in \{0,1\}^m} \psi(h) \prod_{i=1}^{m} A_i^{h_i} (1 - A_i)^{1-h_i}, U_C = \sum_{y \in \{0,1\}^m} \phi(y) \prod_{i=1}^{m} U_i^{y_i} (1 - U_i)^{1-y_i},$$
$$(T_C, K_C) = \left(\max_i T_i + T_J,\ \sum_i K_i + K_J \right). \tag{3}$$

Hence we can reduce every typed guard circuit bottom-up to one equivalent signature.

For exposed parallel-any release, Thm. 4 gives $A_C = 1 - \prod_i (1 - A_i)$ and $U_C = 1 - \prod_i (1 - U_i)$, so redundancy can improve utility while increasing leakage opportunities. These equalities assume independent branches; for unknown dependence, see the sharp marginal-only bounds in Prop. 14.

Answer to puzzle (a): Topology can create feasibility. For a primitive guard with $(a, u, \tau, \kappa) = (.08, .96, .25, 1)$ and requirements $A \le .02$, $U \ge .95$, and $T \le .60$, one guard fails privacy and three in series fail utility and latency. In contrast, two exposed parallel branches followed by one guard give $(A, U, T) \approx (.01, .96, .50)$, satisfying all three requirements (App. B.1).

4.3 How much depth, and how much width?

We return to puzzle (b): given a fixed guard budget, how should it be split between serial depth and parallel width? Let $\mathrm{DW}(k, m; g)$ denote a circuit with k serial stages, each containing m copies of g under the nominal independent-branch model, combined by a zero-overhead collect-all switch under exposed parallel-any tests. By Thm. 4,

$$A_{k,m} = [1 - (1 - a)^m]^k, \qquad U_{k,m} = [1 - (1 - u)^m]^k. \tag{4}$$

Thus added depth lowers both harmful and useful survival, while added width raises both.

A simpler architectural law emerges as guard errors become rare, a regime increasingly plausible as LM guards and verification procedures improve. Consider a family indexed by a reliability parameter λ with $a_g(\lambda) = e^{-\lambda R_g + o(\lambda)}$ and $1 - u_g(\lambda) = e^{-\lambda S_g + o(\lambda)}$, where $R_g, S_g > 0$ are the exponential decay rates of harmful survival and useful-information failure (App. C.2). In this regime, the exact probability calculus becomes the tropical (min–plus) calculus of Tab. 1.

Rates add when all constituent failures are required and take a minimum when any one is sufficient. The electrical analogy is therefore not exact: our privacy resistance follows an add–min law, rather than the max–min law of exponentially scaled physical resistances.

	Privacy R	Task failure S
Series	$R_1 + R_2$	$\min\{S_1, S_2\}$
Parallel-any	$\min_i R_i$	$\sum_i S_i$

Table 1: High-reliability composition laws.

Answer to puzzle (b): depth versus width. For identical guards with privacy and task-failure exponents $r, s > 0$, *depth buys privacy, while width buys task reliability*: $R_{k,m} = kr$ and $S_{k,m} = ms$.

For general switches, the same calculus is governed by the smallest harmful branch set sufficient for leakage and the smallest failed-branch set sufficient for task failure. We call these *privacy certificates* and *task-failure cutsets*, respectively; Thm. 13 gives the full min–plus law.

With a budget of n primitive calls, a natural criterion is to maximize $\min\{kr, ms\}$ subject to $km \le n$. The continuous optimum uses the full budget and balances the rates, $kr = ms$.

Corollary 5 (Optimal depth–width allocation). *Let $r, s > 0$ and let n be the primitive-call budget. In the continuous relaxation, $\max_{km \le n} \min\{kr, ms\} = \sqrt{nrs}$, attained at $k^\star = \sqrt{ns/r}$ and $m^\star = \sqrt{nr/s}$. For integer designs, evaluate $k = 1, \ldots, n$ with $m = \lfloor n/k \rfloor$.*

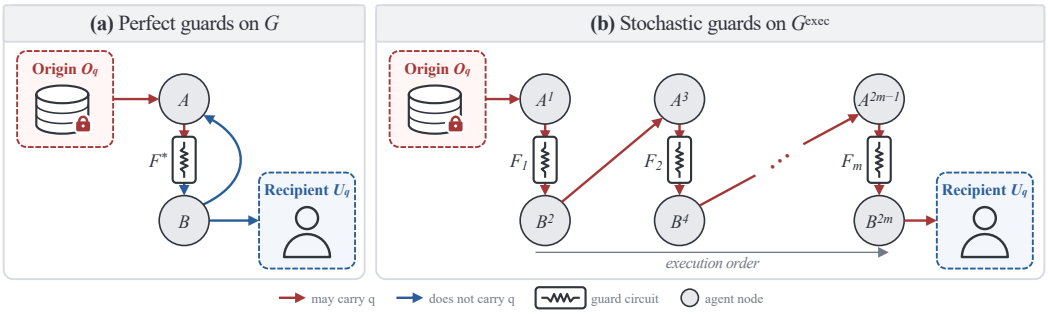

Figure 3: **Perfect versus stochastic guards on a cyclic workflow. (a)** A perfect guard on the minimum cut $F^\star$ blocks every route from O_q to U_q. **(b)** A stochastic guard can leak on each exchange, but unrolling the execution shows it mediates every exchange, forming ordered lines of defense $F_1, \ldots, F_m$ that must all fail for q to reach U_q.

Thus weaker primitive privacy calls for more depth, while weaker primitive task reliability calls for more width. This is a high-reliability max–min criterion; for finite probabilities we instead evaluate Eq. (4). The proof is in App. D.3, and the finite-budget calculation is revisited in App. B.1.

4.4 Circuit Synthesis

The preceding composition laws let us evaluate a fixed circuit; we now use them to choose the circuit itself, formalizing the local design problem in App. C.4. The key structural property is that replacing a subcircuit by a no-worse one cannot hurt any larger allowed circuit. Each circuit C exports a blocking contract $\mathsf{B}_C \subseteq \mathcal{Q}$, the set of protected classes that it is required to block; we define its recursive composition in Eq. (9), App. A.2. For type-compatible circuits, write $C' \preceq C$ if $\mathsf{B}_{C'} \supseteq \mathsf{B}_C$, $A_{C'} \leq A_C$, $U_{C'} \geq U_C$, $T_{C'} \leq T_C$, and $K_{C'} \leq K_C$. Then:

Theorem 6 (Monotone substitution)**.** *Under Asm. 9, let $\mathcal{K}(C_1, \ldots, C_n)$ be a well-typed guard circuit built from the allowed constructors. Then, for every i,*

$$C_i' \preceq C_i \quad \Longrightarrow \quad \mathcal{K}(C_1, \ldots, C_i', \ldots, C_n) \preceq \mathcal{K}(C_1, \ldots, C_i, \ldots, C_n). \tag{5}$$

We prove the result in App. D.5. Together with Thm. 4, it enables exact Pareto synthesis over the stated budgeted series–parallel family: we evaluate candidate circuits recursively and safely prune dominated partial circuits. Our bottom-up dynamic program searches jointly over guard choices and series/parallel/switch topologies while retaining only nondominated states at each size. We give the full optimization problem, dynamic program, complexity analysis, and worked example in Apps. C.4 and C.5 (Alg. 1 and Cor. 17).

5 Where Should We Place Guard Circuits? (Q2)

Having characterized local circuit design, we turn to global placement. A strong circuit on one edge cannot protect an unguarded shortcut, while guarding every edge is unnecessary. We work on the workflow $G = (V, E)$, with origins O_q, unauthorized recipients U_q, and guardable edges $E_g \subseteq E$. As Fig. 3 illustrates, perfect and stochastic guards lead to different placement problems: perfect guards can be studied on the static workflow graph, whereas stochastic guards require reasoning about repeated failures along an execution.

Let $\mathcal{P}_G(O_q, U_q)$ denote the directed paths from O_q to U_q. A feasible separator $F \subseteq E_g$ intersects every such path, equivalently leaving no O_q-to-U_q path after its removal. The formal separator family is given in Eq. (28), App. C.6. For ideal blockers, we show in Thm. 18 that the minimum-cost feasible deployment is exactly a directed minimum cut, a classical object in network flow and security (Ford and Fulkerson, 1956; King et al., 2010; Livshits and Chong, 2013). As shown in Fig. 3(a), a perfect guard need only cover a separating cut rather than every communication edge.

5.1 From One Separator to Several Lines of Defense

Stochastic guards can fail repeatedly: if a guard blocks each exchange with probability .90, then, assuming independent failures across exchanges for illustration, the probability of at least one failure after one hundred exchanges is $1 - .9^{100} \approx 1$. As illustrated in Fig. 3(b), we therefore unroll a finite

execution of the possibly cyclic workflow G into a causal message-flow DAG $G^{\text{exec}} = (V^{\text{exec}}, E^{\text{exec}})$, where repeated invocations of a workflow vertex appear as distinct occurrences. Let O_q^{exec} and U_q^{exec} denote the occurrences of O_q and U_q, respectively. Each execution edge $\tilde{e}$ retains its underlying workflow edge $\pi(\tilde{e}) \in E$, so a guard deployed on $e \in E$ mediates every corresponding execution occurrence. Formal details are in App. C.6. For each guarded execution edge e, let Z_e^{in} and Z_e^{out} denote its pre- and post-mediation messages, and let $\mathcal{H}_e$ denote its pre-edge history. We require

$$\mathbb{P}\big(\mathsf{Rec}_q(Z_e^{\text{out}}) = 1 \mid \mathcal{H}_e\big) \le A_e \, \mathbf{1}\{\mathsf{Rec}_q(Z_e^{\text{in}}) = 1\} \quad \text{a.s.} \tag{6}$$

Thus mediation cannot create the tracked protected information, and A_e upper-bounds its survival probability conditional on the full prior history. An execution separator $\widetilde{F} \subseteq E^{\text{exec}}$ intersects every O_q^{exec}-to-U_q^{exec} path. Pairwise-disjoint execution separators $\widetilde{F}_1, \dots, \widetilde{F}_k$ are *ordered* if every such path crosses them in that order. Let L_q denote the event that some occurrence in U_q^{exec} receives a message from which q is recoverable.

Theorem 7 (Stochastic privacy barriers)**.** *Suppose Eq. (6) holds and, for every $v \notin O_q^{\text{exec}}$ and every outgoing edge $f = (v, w) \in E^{\text{exec}}$ we have,*

$$\mathsf{Rec}_q(Z_f^{\text{in}}) = 1 \implies \exists\, e = (u, v) \in E^{\text{exec}} \text{ such that } \mathsf{Rec}_q(Z_e^{\text{out}}) = 1 \qquad \text{a.s.} \tag{7}$$

If the guarded execution edges contain ordered barriers $\widetilde{F}_1, \dots, \widetilde{F}_k$, then $\mathbb{P}(L_q) \le \prod_{j=1}^{k} \alpha(\widetilde{F}_j) = \exp\Big\{-\sum_{j=1}^{k} \mathcal{R}(\widetilde{F}_j)\Big\}$. Here $\alpha(\widetilde{F}) := \min\{1, \sum_{e \in \widetilde{F}} A_e\}$ and $\mathcal{R}(\widetilde{F}) := -\log \alpha(\widetilde{F})$ is the barrier's privacy resistance.

Condition (7) ensures that non-origin vertices cannot create protected information q; they can only propagate it from incoming messages. We prove the theorem in App. D.8. In particular, if $A_e = a$ and each barrier contains at most m edges with $ma < 1$, then k ordered barriers give $\mathbb{P}(L_q) \le (ma)^k$ (Cor. 20). For a fixed execution graph, the best single-barrier certificate is obtained by a directed minimum cut with edge weights A_e. For a finite single-use DAG this cut is directly on G; for cyclic workflows, a deployed workflow edge may correspond to multiple execution edges through π. More generally, the placement principle is to seek deployments whose executions contain multiple strong ordered barriers. The guarantee applies to finite causal executions, or uniformly to a family of executions when the same certificate holds.

Answer to puzzle (c): beyond minimum cut. With N exposed recipients, a shared guard and m independent recipient-side guards have the same mean exposure, $\mathbb{E}[N] = ma$, but different tails: any exposure has probability a versus $1 - (1-a)^m$, and exposing all m recipients has probability a versus a^m. Mean leakage alone thus does not settle placement; it depends on which exposure events are costly (App. B.1, Thm. 22).

6 Experiments

We test the predictions of Secs. 4 and 5 on agent privacy benchmarks. Experiment 1 (Sec. 6.2) tests whether the composition laws of Secs. 4.2 and 4.3 predict how repeated guards behave in series and in parallel (Q1). Experiment 2 (Sec. 6.3) tests where guards should be placed (Q2): on the minimum cut, and, in a looping workflow, on an edge that every iteration crosses. Full experimental details are provided in Sec. E.

6.1 Experimental setup

Datasets. We use three agent privacy benchmarks: AgentCIBench (ACIB) (Goel and Gurevych, 2026), AgentSocialBench (ASB) (Wang and Jiang, 2026), and PAC-Bench (Park et al., 2026). Each case provides a task, a private input (e.g., financial or medical details), and labels marking which items are protected and which the task requires (Sec. E.1).

Guards. We use a *censoring guard* from the agent-privacy literature, which prompts an LM to reproduce the message with sensitive details deleted (Staab et al., 2025; Jeon et al., 2026); it does not see the protected-item labels. We implement it with six LMs: GPT-5.4, GPT-5-mini, Claude Sonnet 4.6, Claude Haiku 4.5, Gemini 3.5 Flash, and Gemini 3.1 Flash-Lite (Secs. E.2 and E.3). Section E.8 repeats Experiment 1 with an *editing guard*, which replaces sensitive spans with typed placeholders such as `[NAME1]` (Dou et al., 2024; Abdelnabi et al., 2025).

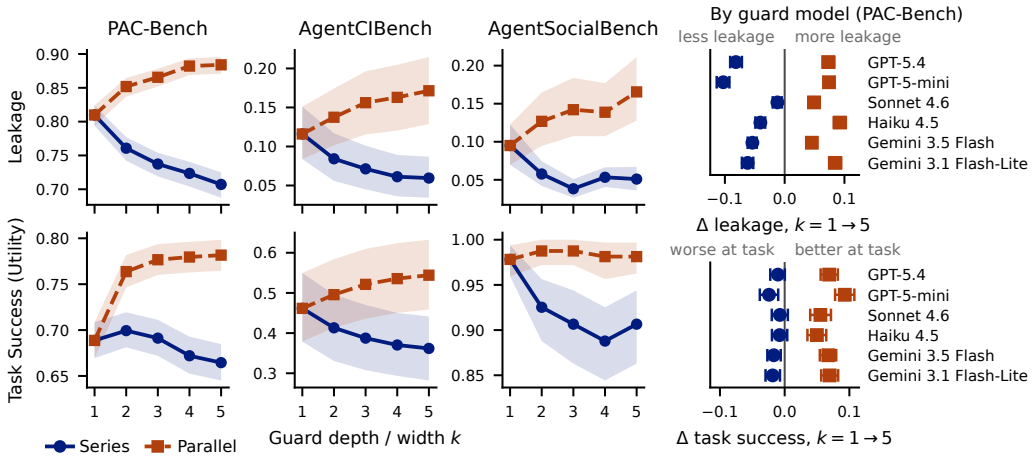

Figure 4: **Series depth reduces leakage; parallel width increases it.** Left: leakage (top) and TS (bottom) against the number of guards k, for the censoring guard with GPT-5-mini. Bands show 95% bootstrap intervals. Right: change in leakage (top) and TS (bottom) from k=1 to k=5 on PAC-Bench for each guard model, with 95% intervals.

Metrics. Leakage A is the fraction of a case's protected items that appear in the released output. Utility U is *Task Success* (TS), which equals 1 if the output satisfies every task requirement. Latency T is the measured wall-clock time of the guard circuit (Sec. E.11). Experiment 2 also reports *Safe Task Success* (STS), which equals 1 if the task succeeds and no protected item is released. ACIB is scored by exact item matching, and ASB and PAC-Bench by a GPT-5.1 judge that follows each benchmark's criteria (Sec. E.6). Significance testing is indicated in each table through bold ($p < 0.05$) or underline ($p < 0.01$), for paired 1-tailed t tests (paired by case) after Holm-Bonferroni correction within each metric. p-values are given in Sec. E.

6.2 Experiment 1: Composing guards (Q1)

We apply the same guard $k \in \{1, \ldots, 5\}$ times in two structures. In *series*, each guard processes the output of the previous one. In *parallel*, k guards process the same input and all of their outputs are released, so a leak from any branch counts. For each case, we run the k=5 circuit once and evaluate each shorter circuit on its first k guards.

As predicted by Eq. (4), series depth reduces leakage and parallel width increases it on all three datasets (Fig. 4). For GPT-5-mini, increasing k from 1 to 5 roughly halves leakage in series on ACIB and ASB and raises it by a similar margin in parallel; on PAC-Bench, leakage is high even at k=1, but the curves diverge in the same directions. At k=5, parallel leaks significantly more than series for every guard model on every dataset (Tab. 2). Sonnet 4.6 shows the weakest effect: its leakage changes little with k in either structure, which suggests that its misses are strongly correlated across repeated calls (Sec. C.3). Task success moves in the opposite direction, again as predicted. Series depth reduces TS (e.g., from 0.46 to 0.36 on ACIB), since each guard can also remove required content, whereas parallel width raises TS on ACIB and PAC-Bench and leaves it near the ceiling on ASB. Neither structure improves both privacy and utility. The editing guard shows the same pattern on ACIB and ASB (Sec. E.8).

	ACIB		ASB		PAC-Bench	
Guard model	Series	Parallel	Series	Parallel	Series	Parallel
GPT-5-mini	.060	**.171**	.051	**.166**	.702	**.880**
GPT-5.4	.114	**.288**	.072	**.120**	.681	**.835**
Haiku 4.5	.033	**.209**	.052	**.085**	.453	**.587**
Sonnet 4.6	.062	**.076**	.055	**.072**	.490	**.551**
Gem. 3.1 FL	.039	**.101**	.056	**.083**	.573	**.719**
Gem. 3.5 F	.096	**.165**	.062	**.171**	.834	**.932**

Table 2: Leakage (fraction of protected items released) at k=5 for the censoring guard, series versus parallel, for every guard model. Bold indicates significance at ($p < 0.05$); underline at ($p < 0.01$).

6.3 Experiment 2: Placing guards (Q2)

Section 5 makes two predictions about where a fixed guard budget should go. On an acyclic workflow, guards belong on the minimum cut (Thm. 18). On a looping workflow, a single guard on an edge that every iteration crosses acts as several ordered barriers (Thm. 7). We test each prediction on a

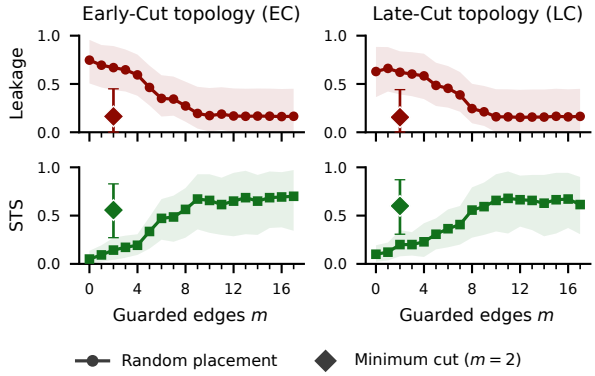

Figure 5: **Guarding the minimum cut reduces leakage and improves STS.** Leakage (top) and STS (bottom) against the number of guarded edges m, for the censoring guard on ACIB. Intervals are 95% bootstrap intervals.

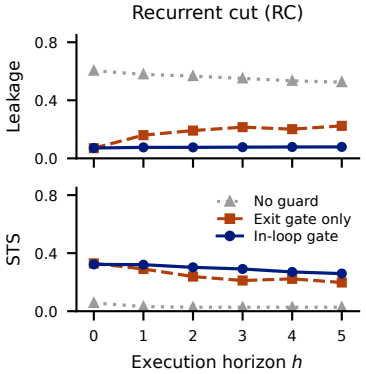

Figure 6: **Recurrence turns one placement into repeated barriers.** Leakage (top) and STS (bottom) against the number of iterations h of a writer–reviewer loop on ACIB.

workflow built to isolate it, using ACIB and GPT-5-mini for all non-guard agents. Since the guards are imperfect, both tests measure empirical placement effects rather than restating the theorems, and we report STS to check that a placement does not reduce leakage by breaking the task.

Minimum cut. We use two acyclic topologies, *Early-Cut* (EC) and *Late-Cut* (LC), which implement a propose–review–resolve workflow with 17 guardable edges (Fig. 8). Each has a unique two-edge minimum cut, between the propose and review stages in EC and between the review and resolve stages in LC. We compare guarding the minimum cut against guarding $m \in \{0, \dots, 17\}$ edges chosen uniformly at random. With two guards, the minimum cut leaks far less than two random edges (GPT-5-mini; EC: 0.16 vs. 0.68; LC: 0.16 vs. 0.61) and roughly matches guarding all 17 edges, whereas random placement needs 6 (EC) and 7 (LC) edges before its leakage is no longer significantly higher (Fig. 5). The minimum cut also raises STS (EC: 0.58 vs. 0.11; LC: 0.60 vs. 0.22). Both differences are significant for every guard model on both topologies (Tab. 3).

	Early-Cut (EC)				Late-Cut (LC)			
	Leakage		STS		Leakage		STS	
Guard model	Cut	Rand	Cut	Rand	Cut	Rand	Cut	Rand
GPT-5-mini	**.164**	.676	**.577**	.114	**.163**	.613	**.600**	.224
GPT-5.4	**.226**	.673	**.550**	.107	**.238**	.593	**.457**	.179
Haiku 4.5	**.205**	.655	**.586**	.114	**.207**	.667	**.379**	.114
Sonnet 4.6	**.164**	.654	**.643**	.150	**.181**	.613	**.550**	.164
Gem. 3.1 FL	**.174**	.687	**.643**	.107	**.189**	.648	**.543**	.150
Gem. 3.5 F	**.161**	.702	**.686**	.079	**.159**	.607	**.571**	.150

Table 3: Minimum cut versus two randomly placed guards on ACIB. Bold indicates significance at ($p < 0.05$).

Recurring cut. A writer agent drafts a recipient-ready message from the private input, a reviewer agent returns feedback, and the loop repeats h times before the writer's final draft is released (Fig. 8). We install one guard either on the writer's outgoing edge, which every iteration crosses (*in-loop*), or on the release edge alone (*exit only*). The two arms coincide at $h{=}0$, so their difference isolates recurrence rather than guard strength (Sec. E.10). We observe that in-loop leakage stays flat as the loop lengthens (0.07 at $h{=}0$ to 0.08 at $h{=}5$), whereas exit-only leakage rises to 0.22, and in-loop STS is higher (0.26 vs. 0.20 at $h{=}5$; Fig. 6 and Tab. 8). Since the private input enters only at the first iteration, every path to the recipient crosses the in-loop guard $h{+}1$ times, which is the condition under which Thm. 7 predicts the barriers to compose.

7 Conclusion

We introduce GuardCircuits, a framework for arranging and placing imperfect guards in multi-agent workflows. Equivalent signatures make guard circuits comparable, the tropical resistance law explains the depth–width tradeoff between privacy and utility, and ordered barriers connect local guard guarantees to end-to-end leakage across causal executions of the workflow. Empirical results on three agent privacy benchmarks support these predictions. Future work should test whether guard signatures calibrated on one workflow transfer to new designs. We discuss limitations in Sec. F.

AI-USE STATEMENT

We used AI tools to revise and edit the writing, to help with literature search and identifying related work, and to support research ideation and parts of the execution, such as drafting code and running experiments. These tools served only as assistants; all AI-assisted content was reviewed by the authors.

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.

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. IPIGuard: A novel tool dependency graph-based defense against indirect prompt injection in LLM agents. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1023–1039, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.53. URL https://aclanthology.org/2025.emnlp-main.53/.

Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. AirGapAgent: Protecting privacy-conscious conversational agents. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 3868–3882, 2024. doi: 10.1145/3658644.3690350.

E. Bassani and I. Sanchez. On guardrail models' robustness to mutations and adversarial attacks. In *Findings of EMNLP 2025*, pages 16995–17006, 2025. doi: 10.18653/v1/2025.findings-emnlp.922.

Z. W. Birnbaum. On the importance of different components in a multicomponent system. pages 581–592, 1969.

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

Yao Dou, Isadora Krsek, Tarek Naous, Anubha Kabra, Sauvik Das, Alan Ritter, and Wei Xu. Reducing privacy risks in online self-disclosures with language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13732–13754, 2024.

Jr. Ford, L. R. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi: 10.4153/CJM-1956-045-5.

Anmol Goel and Iryna Gurevych. Capable but careless: Do computer-use agents follow contextual integrity? *arXiv preprint arXiv:2606.23189*, 2026.

Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S. Yu. The emerged security and privacy of LLM agent: A survey with case studies. *ACM Computing Surveys*, 2025. doi: 10.1145/3773080.

HHS. Health insurance portability and accountability act of 1996. *Public law*, pages 104–191, 1996.

Hyunjun Jeon, Kyuyoung Kim, and Jinwoo Shin. RedacBench: Can AI erase your secrets? *arXiv preprint arXiv:2603.20208*, 2026.

Elliot Kim, Avi Garg, Kenny Peng, and Nikhil Garg. Correlated errors in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *PMLR*, pages 30038–30066, 2025.

D. King, S. Jha, D. Muthukumaran, T. Jaeger, S. Jha, and S. A. Seshia. Automating security mediation placement. In *Programming Languages and Systems (ESOP 2010)*, volume 6012 of *LNCS*, pages 327–344, 2010. doi: 10.1007/978-3-642-11957-6_18.

Seanie Lee, Dong Bok Lee, Dominik Wagner, Minki Kang, Haebin Seong, Tobias Bocklet, Juho Lee, and Sung Ju Hwang. Saferoute: Adaptive model selection for efficient and accurate safety guardrails in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2053–2069, 2025.

H. Li, X. Liu, N. Zhang, and C. Xiao. PIGuard: Prompt injection guardrail via mitigating overdefense for free. In *Proceedings of ACL 2025*, pages 30420–30437, 2025. doi: 10.18653/v1/2025.acl-long.1468.

Kunyang Li, Kyle Domico, Jonathan Gregory, and Patrick McDaniel. MESA: Prioritizing vulnerable communication channels for securing multi-agent systems. *arXiv preprint arXiv:2606.30602*, 2026.

Jinbo Liu, Defu Cao, Yifei Wei, Tianyao Su, Yuan Liang, Yushun Dong, Yan Liu, Yue Zhao, and Xiyang Hu. Topology matters: Measuring memory leakage in multi-agent LLMs. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Findings of the Association for Computational Linguistics: ACL 2026*, pages 39728–39746, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1980. URL https://aclanthology.org/2026.findings-acl.1980/.

B. Livshits and S. Chong. Towards fully automatic placement of security sanitizers and declassifiers. In *Proceedings of POPL*, pages 385–398, 2013. doi: 10.1145/2429069.2429115.

Ian R. McKenzie, Oskar John Hollinsworth, Tom Tseng, Xander Davies, Stephen Casper, Aaron David Tucker, Robert Kirk, and Adam Gleave. STACK: Adversarial attacks on LLM safeguard pipelines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 37728–37737, 2026. doi: 10.1609/aaai.v40i44.41108.

Niloofar Mireshghallah, Hyunwoo Kim, Xuhui Zhou, Yulia Tsvetkov, Maarten Sap, Reza Shokri, and Yejin Choi. Can llms keep a secret? testing privacy implications of language models via contextual integrity theory. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 1892–1915, 2024.

Edward F. Moore and Claude E. Shannon. Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262(3):191–208, 1956. doi: 10.1016/0016-0032(56)90559-2.

Manish Nagireddy, Inkit Padhi, Soumya Ghosh, and Prasanna Sattigeri. When in doubt, cascade: Towards building efficient and capable guardrails. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, volume 8, pages 1812–1821, 2025.

Minjun Park, Donghyun Kim, Hyeonjong Ju, Seungwon Lim, Dongwook Choi, Taeyoon Kwon, Minju Kim, and Jinyoung Yeo. PAC-BENCH: Evaluating multi-agent collaboration under privacy constraints. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Findings of the Association for Computational Linguistics: ACL 2026*, pages 31030–31056, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1552. URL https://aclanthology.org/2026.findings-acl.1552/.

J. S. Provan. The complexity of reliability computations in planar and acyclic graphs. *SIAM Journal on Computing*, 15(3):694–702, 1986. doi: 10.1137/0215050.

A. Satyanarayana and R. K. Wood. A linear-time algorithm for computing K-terminal reliability in series-parallel networks. *SIAM Journal on Computing*, 14(4):818–832, 1985. doi: 10.1137/0214057.

Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. Privacylens: Evaluating privacy norm awareness of language models in action. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 89373–89407. Curran Associates, Inc., 2024. doi: 10.52202/079017-2837.

Robin Staab, Mark Vero, Mislav Balunovic, and Martin Vechev. Language models are advanced anonymizers. In *The Thirteenth International Conference on Learning Representations*, 2025.

Paul Voigt and Axel Von dem Bussche. The EU general data protection regulation (GDPR): A practical guide. 2017.

Prince Zizhuang Wang and Shuli Jiang. AgentSocialBench: Evaluating privacy risks in human-centered agentic social networks. *arXiv preprint arXiv:2604.01487*, 2026.

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. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7261–7276, 2025. doi: 10.18653/v1/2025.acl-long.359.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. Agentleak: A benchmark for internal-channel privacy leakage in multi-agent llm systems. *IEEE Access*, 14:94960–94978, 2026. doi: 10.1109/ACCESS.2026.3704541.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. NetSafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2905–2938, 2025. doi: 10.18653/v1/2025.findings-acl.150.

APPENDIX CONTENTS

A Notation and Formal Setup **15**

A.1 Notation summary . . . 15

A.2 Full workflow specification and blocking contracts . . . 15

A.3 Response laws, nominal model, and circuit events . . . 16

A.4 Parallel selection and fork–join semantics . . . 18

B Worked Examples **19**

B.1 Motivating design puzzles: full calculations . . . 19

B.2 Detailed worked circuit reduction . . . 20

C Additional Theoretical Results **21**

C.1 Evaluation complexity . . . 21

C.2 High-reliability architectural laws . . . 21

C.3 Dependence-robust bounds . . . 22

C.4 Circuit design at a fixed placement . . . 23

C.5 Budgeted topology synthesis . . . 24

C.6 Ideal placement and the homogeneous barrier rule . . . 26

C.7 Recipient-exposure tails . . . 27

D Proofs **28**

D.1 Equivalent signatures . . . 28

D.2 Dual min-plus composition . . . 29

D.3 Balanced depth–width design . . . 33

D.4 Marginal-only fork–join bounds . . . 33

D.5 Monotone substitution . . . 34

D.6 Budgeted topology synthesis . . . 35

D.7 Ideal placement . . . 36

D.8 Stochastic privacy barriers . . . 36

D.9 Centralized versus branch-local exposure . . . 37

E Experiment Details and Additional Results **38**

E.1 Datasets . . . 38

E.2 Guards . . . 38

E.3 Models . . . 39

E.4 Communication topologies and placements . . . 39

E.5 Metrics . . . 39

E.6 Scoring protocol . . . 40

E.7 Per-model composition curves . . . 41

E.8 Editing guard . . . 41

E.9 Full placement results . . . 41

E.10 Recurring cut . . . 41

E.11 Latency 47

F Limitations and Scope **50**

A Notation and Formal Setup

A.1 Notation summary

Symbol	Meaning
$G = (\mathcal{V}, E)$	Directed workflow and deployment graph; nodes are components, edges are possible message channels, and cycles are allowed
$E_g \subseteq E$	Workflow edges on which we may install guard modules
$\mathcal{Z}$	Ambient space of workflow message objects processed by the evaluators and guard circuits
$\mathcal{G}$	Finite library of primitive guard implementations
$\mathcal{Q}$	Protected-information classes
χ_q	Privacy-policy labeling of nodes by $+1$ (authorized source/holder), 0 (neutral), or -1 (unauthorized recipient)
O_q, U_q	Policy-defined sets $\chi_q^{-1}(+1)$ and $\chi_q^{-1}(-1)$
$\mathcal{D}$	Finite set of task demands
$\mathsf{Rec}_q, \mathsf{Use}_d$	Binary recoverability and task-usability evaluators on messages
$d = (s_d, t_d, \mathsf{Use}_d, \eta_d, \bar{T}_d)$	Task demand: source, target, usability evaluator, success threshold, and latency budget
$Z_e^{\mathrm{in}}, Z_e^{\mathrm{out}}$	Messages on edge e immediately before and after guard mediation
B_g	Protected classes that we require module g to block
B_C	Blocking contract exported by circuit C
$\mathfrak{C}(\mathcal{G})$	Guard-circuit family generated from primitive library $\mathcal{G}$
$\mathfrak{C}_q(\mathcal{G})$	Circuits in $\mathfrak{C}(\mathcal{G})$ whose exported contract contains the fixed class q
$(a_g, u_g, \tau_g, \kappa_g)$	Harmful survival, useful survival, latency, and cost of primitive g
$(A(C), U(C), T(C), K(C))$	Equivalent end-to-end circuit signature
$C_1 \circ C_2$	Serial composition
$C_1 \parallel C_2$	Binary exposed parallel-any block $\mathrm{FJ}_{\vee,\vee}(C_1, C_2)$
J	Multi-input selector acting on actual candidate messages; it releases an ordered subset or abstains
ψ, ϕ	Summary-compatible privacy/utility response tables of J; OR and AND are zero–one cases
$\mathrm{FJ}_{\psi,\phi}(C_1, \ldots, C_m)$	Parallel block followed by a summary-compatible selector J; $\mathrm{FJ}_\phi := \mathrm{FJ}_{\vee,\phi}$
G^{exec}	Finite execution-unrolled causal message-flow DAG
ρ	Map from an execution vertex to its underlying workflow component
π	Map from an execution edge to its underlying workflow edge
$\widetilde{F}_j$	Ordered stochastic barrier in G^{exec} induced by the static deployment
A_e	History-conditioned harmful-pass upper bound on execution edge e
$\alpha(\widetilde{F}), \mathcal{R}(\widetilde{F})$	Dependence-robust pass bound and finite privacy resistance of a stochastic barrier
H, Y	Harmful/useful evaluation indicators, not controller inputs
R_C, S_C	Asymptotic privacy and task-failure exponents
$\mathrm{Cert}(\psi), \mathrm{Cut}(\phi)$	Minimal privacy-trigger certificates and task-failure cutsets
N	Exposure count in the fan-out sections; maximum leaf budget in synthesis
$\mathcal{C}_n$	All n-leaf expressions in the declared binary constructor grammar
F_n	Records representing its nondominated contract–signature states, with reconstruction pointers
Ω	Finite library of calibrated binary series or parallel-plus-switch constructors
Prune	Contract–signature dominance pruning within a common state
$\leq_{\mathrm{cx}}$	Convex order

A.2 Full workflow specification and blocking contracts

We distinguish the workflow graph, which specifies communication, from a local guard circuit, which processes a message before it continues. Let $G = (\mathcal{V}, E)$ be a finite directed workflow graph, possibly with cycles, where each edge denotes a possible message channel, and let $E_g \subseteq E$ be the edges on which guards may be installed. Let $\mathcal{Q}$ be a finite set of protected information classes. For each $q \in \mathcal{Q}$,

the privacy policy is a node labeling $\chi_q : \mathcal{V} \to \{+1, 0, -1\}$, with $+1$ denoting an authorized source or holder, 0 a policy-neutral node, and -1 an unauthorized recipient. We define

$$O_q := \{v \in \mathcal{V} : \chi_q(v) = +1\}, \qquad U_q := \{v \in \mathcal{V} : \chi_q(v) = -1\},$$

and write $\mathcal{P}_G(O_q, U_q)$ for the directed paths between these sets.

Let $\mathcal{Z}$ be the ambient space of workflow message objects, including the structured outputs handled by guard circuits. The fixed evaluator $\mathsf{Rec}_q : \mathcal{Z} \to \{0, 1\}$ reports whether protected class q can be recovered from a message, and $\delta_q \in [0, 1]$ bounds the permitted leakage probability.

Let $\mathcal{D}$ be a finite set of task demands. A task demand is

$$d = (s_d, t_d, \mathsf{Use}_d, \eta_d, \bar{T}_d),$$

where $s_d, t_d \in \mathcal{V}$ are the source and target and $\mathsf{Use}_d : \mathcal{Z} \to \{0, 1\}$ reports whether a message contains enough task-relevant information to satisfy demand d. The threshold $\eta_d \in [0, 1]$ is the required task-success probability and $\bar{T}_d \geq 0$ is the latency budget. The same Use_d predicate is applied to intermediate circuit messages when we measure useful-information survival. Shared placement decisions can couple different classes and tasks; each individual circuit or cut result fixes one q and one d.

Let $\mathcal{G}$ be the finite primitive-guard library. Each $g \in \mathcal{G}$ has a declared blocking contract $\mathsf{B}_g \subseteq \mathcal{Q}$ and the operational signature in Def. 1. The library may contain deterministic filters, learned guards, human review, or hybrids. A contract states which classes a guard is required to block; it does not assert zero operational error.

A primitive placement is a map $\gamma : E \to \mathcal{G} \cup \{\bot\}$ with $\gamma(e) = \bot$ for $e \notin E_g$. Here $\bot$ means no guard. The next condition requires every forbidden path to encounter a guard with the appropriate contract.

Definition 8 (Structural admissibility)**.** A placement (G, γ) is *structurally admissible* if

$$\forall q \in \mathcal{Q},\ \forall p \in \mathcal{P}_G(O_q, U_q), \quad \exists e \in p :\ \gamma(e) \neq \bot \ \text{ and } \ q \in \mathsf{B}_{\gamma(e)}. \tag{8}$$

This rules out an unguarded forbidden path, not a guard failure. The same condition applies to composite circuits using their exported contracts.

A.2.1 The contract exported by a circuit

For each q, put

$$b_i^q := \mathbf{1}\{q \notin \mathsf{B}_{C_i}\}, \qquad b^q = (b_1^q, \ldots, b_m^q).$$

If branch contracts are perfectly enforced, any harmful pattern h satisfies $h \leq b^q$. With no creation of an absent harmful mode and the monotone switch law for class q, we export

$$\mathsf{B}_{C_1 \circ C_2} = \mathsf{B}_{C_1} \cup \mathsf{B}_{C_2}, \qquad \mathsf{B}_{\mathrm{FJ}_{\psi,\phi}(C_1,\ldots,C_m)} = \{q \in \mathcal{Q} : \psi(b^q) = 0\}. \tag{9}$$

We suppress the class index on ψ; each class uses its own privacy test and response law. In series, either blocking stage suffices because later guards do not recreate q. At a fork, monotonicity gives

$$0 \leq \psi(h) \leq \psi(b^q) = 0 \qquad (h \leq b^q).$$

Conversely, $\psi(b^q) > 0$ prevents a zero-release guarantee based on the branch contracts alone. Thus Eq. (9) is a conservative contract rule, not a claim that measured leakage is zero. The mechanism or contract must justify any exact zero.

A.3 Response laws, nominal model, and circuit events

The exact calculus needs two kinds of information: what event a module preserves, and when its measured probability can be reused. We specify both before stating the nominal assumptions.

Input and output events. For a circuit C, define

$$H_C^{\text{in}} = \mathsf{Rec}_q(Z_C^{\text{in}}), \quad H_C^{\text{out}} = \mathsf{Rec}_q(Z_C^{\text{out}}),$$
$$Y_C^{\text{in}} = \mathsf{Use}_d(Z_C^{\text{in}}), \quad Y_C^{\text{out}} = \mathsf{Use}_d(Z_C^{\text{out}}).$$

We evaluate harmful and useful survival on their respective input trials:

$$\mathsf{A}(C) := \mathbb{P}(H_C^{\text{out}} = 1 \mid H_C^{\text{in}} = 1), \qquad \mathsf{U}(C) := \mathbb{P}(Y_C^{\text{out}} = 1 \mid Y_C^{\text{in}} = 1). \tag{10}$$

Only positive-probability conditioning events define observed rates. If a mode never reaches a stage, its contribution to the composed rate is zero; no conditional probability on that null event is needed. Let $\mathsf{T}(C)$ and $\mathsf{K}(C)$ be its declared latency and full-execution cost. We write

$$\sigma(C) = \big(\mathsf{A}(C), \mathsf{U}(C), \mathsf{T}(C), \mathsf{K}(C)\big) = (A_C, U_C, T_C, K_C). \tag{11}$$

The notation $A(C), U(C), T(C), K(C)$ denotes the same coordinates.

A switch law on messages. For a supported tuple $z = (z_1, \dots, z_m)$, write $h(z) = (\mathsf{Rec}_q(z_i))_{i=1}^m$ and $y(z) = (\mathsf{Use}_d(z_i))_{i=1}^m$. Let

$$\pi_J(I \mid z) := \mathbb{P}(J \text{ selects } I \mid z), \qquad I \subseteq [m],$$

so $\pi_J(I \mid z) \geq 0$ and $\sum_I \pi_J(I \mid z) = 1$. The output is the ordered subtuple z_I; the empty subtuple denotes no release and has both evaluation labels zero. We call J *summary-compatible* when, for every supported tuple,

$$\sum_{I \subseteq [m]} \pi_J(I \mid z)\mathsf{Rec}_q(z_I) = \psi(h(z)),$$
$$\sum_{I \subseteq [m]} \pi_J(I \mid z)\mathsf{Use}_d(z_I) = \phi(y(z)).$$

These identities imply Eq. (1) by conditioning on a pattern. Pattern-conditioned averages alone do not imply these message-level identities. This distinction matters when upstream message distributions change.

Assumption 9 (Nominal circuit model)**.** Fix (q, d) and the port tests.

(A1) **No creation.** For every primitive guard,

$$H_g^{\text{out}} \leq H_g^{\text{in}}, \qquad Y_g^{\text{out}} \leq Y_g^{\text{in}} \quad \text{a.s.}$$

The inequalities use the usual order on $\{0, 1\}$. Thus a guard may remove a tracked harmful or useful mode, but cannot create one that was absent.

(A2) **Conditional branch independence.** At every fork with branches $C_1, \dots, C_m$, conditional on the tracked mode being present at the fork input, the branch outcomes are independent:

$$\mathbb{P}(H = h \mid H_C^{\text{in}} = 1) = \prod_{i=1}^m A_i^{h_i}(1 - A_i)^{1-h_i},$$
$$\mathbb{P}(Y = y \mid Y_C^{\text{in}} = 1) = \prod_{i=1}^m U_i^{y_i}(1 - U_i)^{1-y_i}.$$

Here A_i, U_i are the branch signatures in that operating regime, and $0^0 = 1$ in the Bernoulli masses. We do not assume independence between H_i and Y_i on the same branch.

(A3) **Stable serial rates.** Let D be a guard or subcircuit placed after any allowed upstream circuit C. Conditional on the tracked mode reaching D, its survival probability is the calibrated rate of D:

$$\mathbb{P}(H_D^{\text{out}} = 1 \mid H_D^{\text{in}} = 1, \text{ upstream circuit } C) = A_D,$$
$$\mathbb{P}(Y_D^{\text{out}} = 1 \mid Y_D^{\text{in}} = 1, \text{ upstream circuit } C) = U_D.$$

Thus the signature of D remains valid when D is composed after any allowed upstream circuit.

(A4) **Stable switch response laws.** For a specified switch J with m branch inputs, let $H = (H_1, \dots, H_m)$ and $Y = (Y_1, \dots, Y_m)$ denote the harmful- and useful-output patterns of its input branches. The switch obeys the response laws defined in Eq. (1):

$$\psi(h) = \mathbb{P}(\mathsf{Rec}_q(Z^{\mathrm{out}}) = 1 \mid H = h), \qquad \phi(y) = \mathbb{P}(\mathsf{Use}_d(Z^{\mathrm{out}}) = 1 \mid Y = y),$$

for $h, y \in \{0,1\}^m$. We assume these response tables remain valid whenever the specified switch is used in an allowed circuit. They are coordinatewise nondecreasing and satisfy

$$\psi(\mathbf{0}) = \phi(\mathbf{0}) = 0.$$

Thus, once the branch outcome pattern is fixed, ψ gives the probability of harmful release and ϕ gives the probability of useful release. Adding harmful branches cannot reduce harmful-release probability, and adding useful branches cannot reduce useful-release probability. Circuit replacements used in synthesis must preserve these switch laws together with (A2) and (A3).

(A5) **Full-execution resources.** Every scheduled branch executes. Serial latency and cost add, while

$$T_{\mathrm{FJ}} = \max_i T_i + T_J, \qquad K_{\mathrm{FJ}} = \sum_i K_i + K_J.$$

The nonnegative overheads T_J, K_J belong to the specified switch. These are synchronous execution bounds; early stopping and input-dependent mean runtimes are outside this model.

Assumptions (A1) and (A3) justify serial multiplication; (A2) and (A4) justify averaging a fixed switch law over product-distributed branch patterns; (A5) supplies the resource coordinates. Separate random seeds do not establish (A2), since branches may share a difficult input. Calibration supports these assumptions only in its declared regime.

For collect-all with exposed-any tests, $\psi(h) = \bigvee_i h_i$ and $\phi(y) = \bigvee_i y_i$. If every branch contribution is required, $\phi(y) = \bigwedge_i y_i$ instead. Uniform random select-one gives $\psi(h) = m^{-1} \sum_i h_i$ and $\phi(y) = m^{-1} \sum_i y_i$. These laws require the stated tuple tests; combining individually nonrevealing entries may invalidate the OR privacy rule. The shorthand $\mathrm{FJ}_\phi = \mathrm{FJ}_{\vee,\phi}$ uses exposed-any privacy; in particular, $C_1 \parallel C_2 = \mathrm{FJ}_{\vee,\vee}(C_1, C_2)$ for the binary exposed parallel-any block.

The two tables describe marginals of one switch, not independent controls. They need not determine joint safe-and-useful release. Candidates stay inside the declared trusted boundary until release; earlier exposure must be evaluated at its own boundary. A content-aware selector that does not satisfy the message-level identities requires richer state or calibration of the entire closed block. Examples appear in App. A.4; Sec. C.3 treats unknown branch dependence.

A.4 Parallel selection and fork–join semantics

The physical components act on messages; the Boolean labels are evaluation summaries. For a common input x, the parallel block and selector implement

$$x \longrightarrow (Z_1, \dots, Z_m) \xrightarrow{J} Z^{\mathrm{out}}, \qquad Z_i \sim C_i(\cdot \mid x). \tag{12}$$

Parallel execution does not imply independence; the factorization in Thm. 4 is the separate assumption in Asm. 9. The candidate tuple remains inside the declared trusted boundary until the selector releases an output.

Selection and filtering are distinct. The selector observes the actual candidate tuple and returns an ordered subtuple $(Z_i)_{i \in I}$ for some $I \subseteq [m]$; $I = \varnothing$ means abstention. It may retain one, several, or all candidates, but does not rewrite them. Redaction, summarization, or generation is represented by a separately calibrated processing module before or after selection.

Summary-compatible response tables. For $z = (z_1, \dots, z_m)$, write $h(z) = (\mathsf{Rec}_q(z_i))_{i=1}^m$ and $y(z) = (\mathsf{Use}_d(z_i))_{i=1}^m$. These are evaluation labels, not required runtime inputs. A switch is summary-compatible when its harmful and useful response probabilities depend on the actual tuple only through these patterns, as defined in App. A.3. Pattern-conditioned averages are reusable component laws only when they remain valid over the admitted upstream message distributions; otherwise we must enrich the state or calibrate the complete parallel-plus-selector block.

Selector	Response tables	Operational meaning
Collect all	$\psi(h) = \mathbf{1}\{\sum_i h_i \geq 1\}$, $\phi(y) = \mathbf{1}\{\sum_i y_i \geq 1\}$	Release the complete ordered tuple; one useful candidate suffices
Uniform select-one	$\psi(h) = m^{-1}\sum_i h_i$, $\phi(y) = m^{-1}\sum_i y_i$	Release one candidate independently of message content
Approval quorum	$\psi(h) = \mathbf{1}\{\sum_i h_i \geq k\}$, $\phi(y) = \mathbf{1}\{\sum_i y_i \geq k\}$	Release a common proposal only when at least k copies survive
Ideal safe selector	$\psi(h) = \mathbf{1}\{\sum_i h_i = m\}$, $\phi(y) = \mathbf{1}\{\sum_i y_i \geq 1\}$	Oracle reference under a domain where every useful candidate is safe
Content-aware selector	Not generally determined by h, y	Enrich the state or calibrate the closed block

Example 10 (Open redundant sanitization)**.** Suppose the selector returns the complete ordered tuple and the output tests report leakage when any entry leaks and task success when any entry preserves the required payload. Then $\psi(h) = \mathbf{1}\{\sum_i h_i \geq 1\}$ and $\phi(y) = \mathbf{1}\{\sum_i y_i \geq 1\}$. Under Asm. 9, $A_{\text{open}} = 1 - \prod_i (1 - A_i)$ and $U_{\text{open}} = 1 - \prod_i (1 - U_i)$. If every branch instead supplies a required contribution, use $\phi(y) = \mathbf{1}\{y_i = 1 \text{ for all } i\}$, giving $U = \prod_i U_i$. If individually nonrevealing entries can jointly reveal the secret, the OR privacy table is invalid and richer privacy state is required.

Example 11 (Summary-compatible stochastic selector)**.** For two candidates, suppose the selector releases the harmful candidate with probability ϵ whenever exactly one candidate is harmful, and releases the useful candidate with probability α whenever exactly one is useful. Assuming exchangeable branch positions, $\psi(1,0) = \psi(0,1) = \epsilon$, $\psi(1,1) = 1$, and $\phi(1,0) = \phi(0,1) = \alpha$, $\phi(1,1) = 1$, with zero response at $(0,0)$. Under (A2),

$$A_J = A_1 A_2 + \epsilon[A_1(1 - A_2) + (1 - A_1)A_2], \tag{13}$$

$$U_J = U_1 U_2 + \alpha[U_1(1 - U_2) + (1 - U_1)U_2]. \tag{14}$$

The same executable policy must realize both tables. For fixed $\epsilon > 0$, singleton harmful patterns determine the privacy exponent; if $\epsilon = 0$, only joint harmful survival contributes. A reliability-dependent $\epsilon(\lambda)$ is outside the fixed-table exponent theorem.

Abstention and resource overhead. A selector may abstain even when every branch is useful, so $\phi(\mathbf{1}) < 1$ is allowed in the finite calculus. The rare-error theorem separately requires $\phi(\mathbf{1}) = 1$ to avoid a nonvanishing task-failure floor. Selector latency and cost are charged as (T_J, K_J); subsequent redaction or generation is charged through its own calibrated module.

B Worked Examples

B.1 Motivating design puzzles: full calculations

Each puzzle asks a system-level question that an isolated guard benchmark cannot answer. Numerical values below are rounded to two decimal places; very small probabilities are shown in scientific notation.

Puzzle (a): why can a mixed circuit beat both one guard and a deep cascade? Take identical guards $(a, u, \tau, \kappa) = (.08, .96, .25, 1)$ and require $\mathsf{A}(C) \leq .02$, $\mathsf{U}(C) \geq .95$, and $\mathsf{T}(C) \leq .60$. The three representative designs are:

Design	A	U	T
One guard	.08	.96	.25
Three in series	5.1×10^{-4}	.88	.75
Two parallel-any, then one serial	.01	.96	.50

One guard fails privacy; the deep cascade fails utility and latency; the mixed circuit satisfies all three requirements. The point is not that the mixed circuit is universally best, but that topology changes which tradeoffs are feasible.

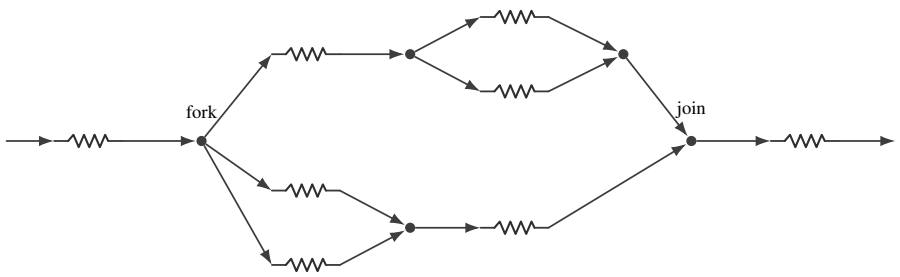

Figure 7: A nested series/fork–join circuit. Bottom-up reduction replaces each subcircuit by its equivalent signature.

Puzzle (b): with a fixed guard budget, should we buy depth or width? For k serial stages of width m, the circuit uses km primitive calls. Under exposed parallel-any semantics, $A_{k,m} = [1-(1-a)^m]^k$ and $U_{k,m} = [1-(1-u)^m]^k$. Thus depth reduces both harmful and useful survival, whereas width increases both. In the high-reliability regime, Thm. 13 and Cor. 5 show that depth contributes additively to privacy resistance while width contributes additively to task-failure resistance.

Puzzle (c): one shared guard or one guard per recipient? For three recipients with $a = .08$ and $u = .96$, let N be the number that receive forbidden content. Then:

Placement	$\mathbb{E}[N]$	$\Pr(N \geq 1)$	$\Pr(N = 3)$	all three useful
Shared upstream	.24	.08	.08	.96
Independent branches	.24	.22	5.1×10^{-4}	.88

The mean exposure is identical, but the tail events rank the placements in opposite directions. Theorem 22 gives the general convex-order result.

B.2 Detailed worked circuit reduction

The first use of the calculus is analytical: we can reduce a nested guard circuit to one signature and check it against system constraints. The same reduction also shows why a locally stronger guard need not improve the complete design.

Consider

$$\begin{aligned} C &= G_0 \circ (C_L \parallel C_R) \circ G_7, \\ C_L &= G_1 \circ (G_3 \parallel G_4), \\ C_R &= (G_2 \parallel G_5) \circ G_6. \end{aligned} \tag{15}$$

We use zero-overhead collect-all switches and the nominal model. The primitive signatures are:

g	a_g	u_g	τ_g	κ_g
G_0	.30	.990	.10	.20
G_1	.12	.960	.20	.50
G_2	.08	.940	.22	.60
G_3	.10	.970	.15	.40
G_4	.18	.990	.08	.20
G_5	.20	.995	.05	.10
G_6	.15	.980	.12	.30
G_7	.10	.990	.10	.25

Bottom-up reduction gives the following signatures. Calculations use the unrounded component values; displayed results are rounded to two decimals, with scientific notation below .01.

Module	A	U	T	K
$G_3 \parallel G_4$	.26	1.00	.15	.60
C_L	.03	.96	.35	1.10
$G_2 \parallel G_5$	.26	1.00	.22	.70
C_R	.04	.98	.34	1.00
$C_L \parallel C_R$	.07	1.00	.35	2.10
$G_0 \circ (C_L \parallel C_R)$	.02	.99	.45	2.30
C	$\mathbf{2.1 \times 10^{-3}}$	**.98**	**.55**	**2.55**

Using the unrounded values, the full circuit satisfies $A(C) \le .005$, $U(C) \ge .975$, and $T(C) \le .60$. Removing G_7 gives $\sigma(C \setminus G_7) \approx (.02, .99, .45, 2.30)$ and fails privacy. Replacing it by $G_7' = (.04, .985, .16, .45)$ gives $\sigma(C[G_7']) \approx (8.4 \times 10^{-4}, .97, .61, 2.75)$, which fails useful-survival and latency. Hence a locally stronger privacy filter is not necessarily a better system design.

C Additional Theoretical Results

Before turning to the high-reliability laws, we briefly discuss the computational cost of evaluating a fixed circuit from its component signatures. We then study the asymptotic regime and dependence-robust bounds.

C.1 Evaluation complexity

We first ask how expensive it is to evaluate a fixed guard-circuit expression, once its topology, primitive signatures, and switch response tables are given. Using Thm. 4, we compute the equivalent signature $\sigma(C)$ bottom-up and analyze the cost of this reduction. A postorder traversal visits each expression node once. At a fork with m branches, recursively extending the Bernoulli masses requires $2 + 4 + \cdots + 2^m = O(2^m)$ operations. Summing the two table-weighted responses has the same cost. Recomputing a length-m product separately for each pattern instead costs $O(m2^m)$.

OR and AND use $O(m)$ product formulas. For a threshold, let p_i be the branch probabilities and let $b_j(r)$ be the probability of r successes among the first j branches. Starting from $b_0(0) = 1$ and zero outside $0 \le r \le j$, use

$$b_j(r) = (1 - p_j)b_{j-1}(r) + p_j b_{j-1}(r - 1).$$

There are $\sum_{j=1}^{m}(j + 1) = O(m^2)$ states and constant work per state; sum $b_m(r)$ over the accepted counts. Thus a circuit with bounded fan-in has linear evaluation cost in its number of expression nodes, with constants determined by the fixed switch tables.

C.2 High-reliability architectural laws

The finite calculus evaluates a particular implementation. Here we ask which failure patterns dominate as guards become more reliable. The parameter λ indexes a family in which both harmful pass and task failure vanish. It need not be model size or a rejection threshold: tightening a threshold can improve privacy while reducing utility.

Definition 12 (Rare-error exponents)**.** For a primitive family g_λ, put $A_g(\lambda) = \mathsf{A}(g_\lambda)$ and $U_g(\lambda) = \mathsf{U}(g_\lambda)$. Assume

$$r_g := \lim_{\lambda\to\infty} -\frac{1}{\lambda}\log A_g(\lambda), \qquad s_g := \lim_{\lambda\to\infty} -\frac{1}{\lambda}\log(1 - U_g(\lambda)) \tag{16}$$

exist in $(0, \infty]$, with $-\log 0 = +\infty$. For a fixed circuit expression C, with fixed port tests and switch tables, replace each leaf by its indexed implementation to obtain C_λ, and define, when the limits exist,

$$R_C := \lim_{\lambda\to\infty} -\frac{1}{\lambda}\log \mathsf{A}(C_\lambda), \qquad S_C := \lim_{\lambda\to\infty} -\frac{1}{\lambda}\log(1 - \mathsf{U}(C_\lambda)). \tag{17}$$

Larger exponents mean faster error decay. For finite r_g, s_g, Eq. (16) is equivalent to $A_g(\lambda) = e^{-\lambda r_g + o(\lambda)}$ and $1 - U_g(\lambda) = e^{-\lambda s_g + o(\lambda)}$. These are rates of logarithmic error decay, not finite electrical resistances.

Minimal failure sets. Write $[m] = \{1, \dots, m\}$ and let $\mathbf{1}_I$ be the indicator of $I \subseteq [m]$. Define

$$\mathrm{Cert}(\psi) = \{L \subseteq [m] : \psi(\mathbf{1}_L) > 0, \quad \psi(\mathbf{1}_K) = 0 \,\forall K \subsetneq L\},$$
$$\mathrm{Cut}(\phi) = \{I \subseteq [m] : \phi(\mathbf{1} - \mathbf{1}_I) < 1, \quad \phi(\mathbf{1} - \mathbf{1}_K) = 1 \,\forall K \subsetneq I\}.$$

A privacy certificate is a smallest set of harmful branches that permits leakage with positive probability. A task-failure cutset is a smallest set of failed branches that permits task failure with positive probability. For Boolean tables these events are forced; for fractional tables they need only be possible. These are sets of branch indices, not cuts in the workflow graph. We use $\min \varnothing = +\infty$.

Theorem 13 (Dual min-plus composition law)**.** *Suppose Asm. 9 holds for every λ, and let $C_{1,\lambda}, \dots, C_{m,\lambda}$ be circuit families as in Def. 12, with $\phi(\mathbf{1}) = 1$ at every switch. Then the circuit exponents exist and satisfy*

$$R_{C_1 \circ C_2} = R_1 + R_2, \qquad S_{C_1 \circ C_2} = \min\{S_1, S_2\}, \tag{18}$$

$$R_{\mathrm{FJ}_{\psi,\phi}(C_1,\dots,C_m)} = \min_{L \in \mathrm{Cert}(\psi)} \sum_{i \in L} R_i, \qquad S_{\mathrm{FJ}_{\psi,\phi}(C_1,\dots,C_m)} = \min_{I \in \mathrm{Cut}(\phi)} \sum_{i \in I} S_i. \tag{19}$$

Special cases. Consequently, exposed-any privacy $\psi = \vee$ gives $R = \min_i R_i$, whereas $\psi = \wedge$ gives $R = \sum_i R_i$. Task-any $\phi = \vee$ gives $S = \sum_i S_i$, whereas task-all $\phi = \wedge$ gives $S = \min_i S_i$. An at-least-ℓ privacy threshold has exponent equal to the sum of the ℓ smallest R_i; an at-least-k-of-m task threshold has failure exponent equal to the sum of the $m - k + 1$ smallest S_i. The relevant minimal sets have exactly those sizes. The full proof is in App. D.2.

Composition	Privacy resistance R	Task-failure resistance S
Series	sum	minimum
General fork–join	minimum certificate weight	minimum failure-cut weight
Exposed parallel-any	minimum	sum

This explains the dual roles of depth and width: serial privacy failure requires survival through every stage, while exposed parallel privacy failure needs only one harmful branch. Task failure reverses these requirements. The conclusion concerns fixed architectures as $\lambda \to \infty$; it does not assert a uniform approximation for circuit sizes growing with λ.

A switch can impose an error floor. If $\phi(\mathbf{1}) < 1$ is fixed and $U_i(\lambda) \to 1$, then the branch success vector converges in probability to $\mathbf{1}$. Therefore $U_C(\lambda) \to \phi(\mathbf{1})$ and $S_C = 0$. This is why the theorem requires $\phi(\mathbf{1}) = 1$. The finite law still applies when this condition fails or when a switch table depends on λ.

C.3 Dependence-robust bounds

Parallel LM guards may share model weights, prompts, providers, or retrieved context, so we must test branch independence rather than presume it. If only branch marginals are known, the following bounds are sharp.

Proposition 14 (Marginal-only fork–join bounds)**.** *For harmful branch events with probabilities A_i under exposed-any privacy $\psi = \vee$,*

$$\max_i A_i \le A_{\mathrm{FJ}} \le \min\left\{1, \sum_i A_i\right\}. \tag{20}$$

For parallel-any useful success,

$$\max_i U_i \le U_\vee \le \min\left\{1, \sum_i U_i\right\}, \tag{21}$$

and for parallel-all,

$$\max\left\{0, \sum_i U_i - (m-1)\right\} \le U_\wedge \le \min_i U_i. \tag{22}$$

Each bound is attainable by some joint distribution with the stated marginals.

The proof in App. D.4 constructs dependence structures that attain every endpoint, so the bounds in Prop. 14 are sharp. These closed-form expressions apply to the OR/AND response laws above. The same idea extends to an arbitrary fixed response table $\rho : \{0,1\}^m \to [0,1]$. Let $X = (X_1, \dots, X_m) \in \{0,1\}^m$ denote the relevant branch outcome vector: $X = H$ for privacy and $X = Y$ for utility. Write $p_i = \mathbb{P}(X_i = 1)$ and $w_x = \mathbb{P}(X = x)$ for $x \in \{0,1\}^m$. If only the branch marginals $p_i = \mathbb{P}(X_i = 1)$ are known, then all compatible dependence structures are exactly the vectors $w = (w_x)$ satisfying

$$w_x \geq 0, \qquad \sum_x w_x = 1, \qquad \sum_{x:x_i=1} w_x = p_i \quad (i \in [m]).$$

For a given joint law w, the switch response probability is $\sum_x \rho(x) w_x$ by the law of total probability. Hence, provided the same response table ρ remains valid across the couplings considered, the sharp lower and upper response probabilities are obtained by minimizing and maximizing $\sum_{x \in \{0,1\}^m} \rho(x) w_x$ over the constraints above. Thus Prop. 14 gives closed-form solutions for important OR/AND cases, while this finite linear program handles a general fixed response table. Additional message-level restrictions may exclude some couplings from the marginal event model.

Conditional serial bounds. A similar robust statement is available for serial composition without assuming history-independent stage behavior. Let H_i be the harmful-mode indicator after stage i, and let $\mathcal{H}_i$ contain the input and preceding history. If $H_i \leq H_{i-1}$ a.s. and $\mathbb{P}(H_i = 1 \mid \mathcal{H}_i) \leq \bar{a}_i H_{i-1}$, then iterated expectation gives $\mathbb{P}(H_k = 1) \leq \mathbb{P}(H_0 = 1) \prod_{i=1}^k \bar{a}_i$. Likewise, if $\mathbb{P}(Y_i = 1 \mid \mathcal{H}_i) \geq \underline{u}_i Y_{i-1}$, then $\mathbb{P}(Y_k = 1) \geq \mathbb{P}(Y_0 = 1) \prod_{i=1}^k \underline{u}_i$. These are robust inequalities rather than the nominal equalities of the composition law, and require no unconditional independence across serial stages.

C.4 Circuit design at a fixed placement

Fix one protected class q and one task demand d, and let $e \in E_g$ be a guardable edge selected for protection of q.

Definition 15 (Admissible circuit-design family)**.** Let $\mathfrak{C}(\mathcal{G})$ denote the guard circuits of Def. 3 whose primitive leaves lie in $\mathcal{G}$. We define

$$\mathfrak{C}_q(\mathcal{G}) := \{C \in \mathfrak{C}(\mathcal{G}) : q \in \mathsf{B}_C\}.$$

For fixed local design targets $\bar{A} \in [0,1]$, $\underline{U} \in [0,1]$, and $\bar{T} \geq 0$, the local synthesis problem is

$$\begin{aligned} C^\star_{q,d} \in \operatorname*{arg\,min}_{C \in \mathfrak{C}_q(\mathcal{G})} \quad & K_C \\ \text{s.t.} \quad & A_C \leq \bar{A}, \quad U_C \geq \underline{U}, \quad T_C \leq \bar{T}. \end{aligned} \tag{23}$$

These local targets are supplied by the enclosing workflow design and need not equal the end-to-end privacy, utility, or latency requirements. Appendix C.5 solves this problem over the budgeted family $\bigcup_{n=1}^N \mathcal{C}_n$ generated by at most N primitive occurrences and the constructor library Ω.

The monotone-substitution result used for Pareto pruning is stated in Thm. 6 and proved in App. D.5.

C.4.1 Pareto frontier

A useful circuit need not minimize every coordinate. We retain a design unless another attainable design is no worse in all four coordinates.

Definition 16 (Pareto front)**.** For $\sigma = (A, U, T, K)$, put $\boldsymbol{\ell}(\sigma) = (A, 1-U, T, K)$. For vectors, $x \preceq y$ means $x_j \leq y_j$ for every j, and $x \prec y$ additionally requires $x \neq y$. Define

$$\mathrm{PF}(S) = \{x \in S : \nexists y \in S \text{ with } y \prec x\}.$$

The achievable loss frontier for the fixed class q is

$$\mathrm{PF}\{\boldsymbol{\ell}(\sigma(C)) : C \in \mathfrak{C}_q(\mathcal{G})\}. \tag{24}$$

C.5 BUDGETED TOPOLOGY SYNTHESIS

We now choose both the guards and their arrangement for the fixed protected class q. The input is the primitive library $\mathcal{G}$, an integer leaf budget $N \geq 1$, and a finite set Ω of binary constructors. A constructor is either series or a parallel block followed by a specified switch J, with fixed realizable tables and overheads. Thus the search may choose among switches in the library; it does not optimize arbitrary response functions.

Search family. All states share the same boundary types, port tests, and allowed calibration regime, and Asm. 9 holds for all compositions considered. The numerical signature remains for the fixed pair (q, d), while each partial circuit also carries its exported contract B_C. If the remaining attributes differ, they must index separate frontiers; dominance alone cannot justify pruning across them.

Let $\mathcal{C}_n$ be all circuit expressions with exactly n primitive occurrences, generated by

$$\mathcal{C}_1 = \mathcal{G}, \qquad \mathcal{C}_n = \bigcup_{i=1}^{n-1} \bigcup_{\omega \in \Omega} \{\omega(C_L, C_R) : C_L \in \mathcal{C}_i,\ C_R \in \mathcal{C}_{n-i}\}.$$

Each occurrence is a separate invocation, even when two leaves use the same guard type. Switches do not count as primitive leaves, but their latency and cost enter the signature. This is the declared binary circuit family, not arbitrary cross-linked graphs or arbitrary multi-input switches.

Build, combine, prune. A record (B, σ, θ) stores the exported contract B, the signature $\sigma = (A, U, T, K)$, and a back-pointer θ to a primitive or to a constructor and its two child records. For contract–signature states, write

$$(B', \sigma') \preceq_{\mathrm{cs}} (B, \sigma) \quad \Longleftrightarrow \quad B' \supseteq B \ \text{ and } \ \boldsymbol{\ell}(\sigma') \preceq \boldsymbol{\ell}(\sigma),$$

and write $\prec_{\mathrm{cs}}$ when the dominance is strict. The operator Prune removes strictly dominated records and keeps one representative of each remaining equal contract–signature state. This is precisely the order used by Thm. 6, projected to the stored state.

Initialize

$$F_1 = \mathrm{Prune}\{(\mathsf{B}_g, \sigma(g), g) : g \in \mathcal{G}\}, \tag{25}$$

and compute, for $n \geq 2$,

$$F_n = \mathrm{Prune}\left(\bigcup_{i=1}^{n-1} \bigcup_{\omega \in \Omega} \omega(F_i, F_{n-i}) \right). \tag{26}$$

Here $\omega(F_i, F_j)$ means: combine every pair of records, compute the parent contract from Eq. (9) and the parent signature from Eqs. (2) and (3), and retain its child back-pointers. The split $n = i + (n-i)$ assigns i leaves to the left child and $n - i$ to the right child; it does not split a message or a guard. Every binary expression has one such root split. By Thm. 6, a dominated partial circuit cannot become preferable after further allowed composition.

Algorithm 1 Exact budgeted guard-circuit synthesis

Require: $\mathcal{G}$, $N \geq 1$, Ω satisfying the common-state assumptions above
Ensure: One representative per nondominated contract–signature state with at most N leaves

$F_1 \leftarrow \mathrm{Prune}\{(\mathsf{B}_g, \sigma(g), g) : g \in \mathcal{G}\}$
if $F_1 = \varnothing$ or $\Omega = \varnothing$ **then**
  return F_1 ▷ no larger expression can be formed
end if
for $n = 2, \ldots, N$ **do**
  $S_n \leftarrow \varnothing$
  for $i = 1, \ldots, n-1$ **do** ▷ i left leaves, $n-i$ right leaves
    for all $\omega \in \Omega$ **do**
      for all $(B_L, \sigma_L, \theta_L) \in F_i$ and $(B_R, \sigma_R, \theta_R) \in F_{n-i}$ **do**
        Compute B from B_L, B_R using Eq. (9)
        Compute σ from σ_L, σ_R using Eqs. (2) and (3)
        $\theta \leftarrow (\omega, \theta_L, \theta_R)$ ▷ store pointers, not copied trees
        Add (B, σ, θ) to S_n
      end for
    end for
  end for
  $F_n \leftarrow \mathrm{Prune}(S_n)$ ▷ discard dominated records and merge ties
end for
return $\mathrm{Prune}\left(\bigcup_{n=1}^{N} F_n\right)$

Corollary 17 (Budgeted topology synthesis)**.** *Under the search-family assumptions above, for every* $1 \leq n \leq N$,

$$\{(B,\sigma) : (B,\sigma,\theta) \in F_n\} = \{(\mathsf{B}_C, \sigma(C)) : C \in \mathcal{C}_n,\ \nexists D \in \mathcal{C}_n \text{ with } (\mathsf{B}_D, \sigma(D)) \prec_{\mathrm{cs}} (\mathsf{B}_C, \sigma(C))\}.$$

Moreover,

$$\{(B,\sigma) : (B,\sigma,\theta) \in \mathrm{Prune}\left(\bigcup_{n=1}^{N} F_n\right)\}$$
$$= \left\{(\mathsf{B}_C, \sigma(C)) : C \in \bigcup_{n=1}^{N} \mathcal{C}_n,\ \nexists D \in \bigcup_{n=1}^{N} \mathcal{C}_n \text{ with } (\mathsf{B}_D, \sigma(D)) \prec_{\mathrm{cs}} (\mathsf{B}_C, \sigma(C))\right\}.$$

Thus F_n is exactly the nondominated contract–signature frontier for n-leaf circuits, and pruning $\bigcup_{n=1}^{N} F_n$ gives the corresponding frontier over all circuits with at most N leaves. To solve the fixed-q synthesis problem in Eq. (23), retain records with $q \in B$, apply $A \leq \bar{A}$, $U \geq \underline{U}$, and $T \leq \bar{T}$, and then choose minimum K. Do not impose these root performance thresholds on partial circuits: a later composition can make them feasible. The proof is in App. D.6.

Complexity. We count arithmetic operations and comparisons, treating $\mathcal{Q}$ and the binary response tables as fixed-size; contract inclusion and signature dominance then cost $O(1)$. The empty-library cases return F_1; define $F_n = \varnothing$ for $n \geq 2$ in those cases. Otherwise, at size n there are $n-1$ root splits. Split $i + (n-i)$ combines $|F_i||F_{n-i}|$ pairs for each constructor, so the candidate count is $M_n = |\Omega| \sum_{i=1}^{n-1} |F_i||F_{n-i}|$. Each candidate requires a constant-size contract/signature computation and two back-pointers. Consequently the total number of compositions is

$$\sum_{n=2}^{N} M_n = |\Omega| \sum_{n=2}^{N} \sum_{i=1}^{n-1} |F_i|\,|F_{n-i}|. \tag{27}$$

If $|F_i| \leq F$, the $N(N-1)/2$ splits give $O(|\Omega| N^2 F^2)$ compositions. This is not the total pruning time. A direct implementation of Prune compares every candidate pair. Set $M_1 = |\mathcal{G}|$ and $M_{\mathrm{out}} = \sum_{n=1}^{N} |F_n|$. The total running time is

$$O\left(|\mathcal{G}|^2 + \sum_{n=2}^{N} (M_n + M_n^2) + M_{\mathrm{out}}^2\right).$$

Since each dominance test costs $O(1)$ under the fixed-$\mathcal{Q}$ convention and $M_n \le |\Omega|(n-1)F^2$, this direct bound is $O(|\mathcal{G}|^2 + |\Omega|^2 N^3 F^4 + N^2 F^2)$ for nonempty Ω. Storing the frontiers and one candidate batch needs $O(\sum_n |F_n| + \max_n M_n)$ records; reconstructing one selected n-leaf circuit takes $O(n)$ time.

These bounds exclude the cost of model calibration and the bit cost of exact arithmetic. The procedure is exact but not polynomial in the worst case: Pareto frontiers may grow exponentially with the leaf budget. Its advantage over exhaustive enumeration is that subproblems are reused and every dominated partial circuit can be discarded without loss of optimality. In favorable regimes where $|F_n| \le F$ for all $n \le N$, the direct implementation runs in $O(|\mathcal{G}|^2 + |\Omega|^2 N^3 F^4 + N^2 F^2)$ time; in particular, for fixed F, Ω, and $\mathcal{G}$, this is $O(N^3)$.

C.5.1 A PRUNING EXAMPLE

Suppose $\mathcal{G}$ contains three primitive guards with the same contract $B = \{q\}$ and signatures

$$\sigma(g_1) = (.05, .80, 1, 1), \qquad \sigma(g_2) = (.40, .99, 1, 1), \qquad \sigma(g_3) = (.15, .88, 1.1, 1.1).$$

No primitive dominates another: g_1 has better privacy but worse utility than the others, while g_2 has the best utility but worst privacy. Hence all three states survive in F_1.

Now consider two-leaf series circuits. By Thm. 4,

$$\sigma(g_1 \circ g_2) = (.020, .792, 2, 2), \qquad \sigma(g_3 \circ g_3) = (.023, .77, 2.2, 2.2).$$

Since their contracts are equal and the first circuit is no worse in every signature coordinate,

$$x(g_1 \circ g_2) \prec_{\mathrm{cs}} x(g_3 \circ g_3).$$

Thus $g_3 \circ g_3$ is removed when constructing F_2, even though g_3 was not dominated in F_1. By Thm. 6, no later composition containing $g_3 \circ g_3$ needs to be retained: replacing that subcircuit by $g_1 \circ g_2$ produces a no-worse complete circuit.

This illustrates the role of the dynamic program: dominance can emerge only after composition, and once it does, the dominated partial design and all of its future extensions can be discarded without loss of optimality.

C.6 IDEAL PLACEMENT AND THE HOMOGENEOUS BARRIER RULE

Fix one protected class q and placement costs $c_e \ge 0$ on the guardable edges $E_g \subseteq E$. For $F \subseteq E_g$, let $c(F) = \sum_{e \in F} c_e$, and define

$$\mathfrak{F}_q(G) := \{F \subseteq E_g : \mathcal{P}_{G-F}(O_q, U_q) = \varnothing\}. \tag{28}$$

Thus F is feasible exactly when removing its edges destroys every forbidden O_q-to-U_q path.

Theorem 18 (Ideal placement)**.** *Let $\widehat{G}_q$ be the augmented graph obtained from G by adding a super-source s_q, a super-sink t_q, edges from s_q to o for every $o \in O_q$, and edges from u to t_q for every $u \in U_q$. Assign capacity c_e to every guardable edge $e \in E_g$, and capacity M_q to every other edge, where $M_q > \sum_{e \in E_g} c_e$. If $\mathfrak{F}_q(G) \neq \varnothing$, then*

$$\min_{F \in \mathfrak{F}_q(G)} c(F) = \lambda_{s_q t_q}(\widehat{G}_q), \tag{29}$$

where $\lambda_{s_q t_q}(\widehat{G}_q)$ denotes the minimum directed s_q-t_q cut value.

We give the proof in App. D.7. Only original guardable edges can lie in an optimal finite-capacity cut, because every other edge has capacity larger than the cost of any feasible guarded separator.

Ordered barriers and edge histories. During a finite execution, the static workflow G unfolds into vertex occurrences. We write v^t for an occurrence of workflow vertex $v \in \mathcal{V}$ and define $\rho(v^t) = v$. If a message sent by v^t is received or used by $u^{t'}$, then $v^t \prec u^{t'}$. These dependencies induce a partial causal order on the execution.

Definition 19 (Execution-unrolled message-flow graph)**.** For a finite execution of $G = (\mathcal{V}, E)$, let V^{exec} contain all vertex occurrences v^t, and let E^{exec} contain $(v^t, u^{t'})$ whenever the execution transmits a message from v^t to $u^{t'}$, with $(v, u) \in E$. We call $G^{\text{exec}} = (V^{\text{exec}}, E^{\text{exec}})$ the *execution-unrolled message-flow graph*.

Every execution edge follows the strict causal order $\prec$; hence a directed cycle would imply $v^t \prec v^t$ for some occurrence v^t, contradicting irreflexivity. Thus G^{exec} is a DAG. Repeated executions of the same workflow vertex appear as distinct vertices, while concurrent occurrences need not be ordered. Each execution edge $(v^t, u^{t'})$ corresponds to the workflow edge $(v, u) \in E$; define $\pi(v^t, u^{t'}) = (v, u)$. Thus a guard deployed on $e \in E$ mediates every execution edge $\tilde{e}$ with $\pi(\tilde{e}) = e$.

Set $O_q^{\text{exec}} = \rho^{-1}(O_q)$ and $U_q^{\text{exec}} = \rho^{-1}(U_q)$. If G is already a finite single-use DAG, we may take $G^{\text{exec}} = G$, with ρ and π equal to the identity maps. A missing transmission has a nonrevealing output. The event L_q means that some occurrence in U_q^{exec} receives a message with $\mathsf{Rec}_q = 1$, not that several individually nonrevealing messages jointly disclose q.

Feasible execution separators $\widetilde{F}_1, \dots, \widetilde{F}_k$ are *ordered* if they are pairwise disjoint and, on every forbidden path $p = (e_1, \dots, e_\ell)$ in G^{exec}, there are indices

$$1 \le t_1 < \cdots < t_k \le \ell, \qquad e_{t_j} \in \widetilde{F}_j.$$

Thus every forbidden causal route must cross all selected lines of defense in order. Each guarded execution edge inherits its deployed module through the map $\pi : E^{\text{exec}} \to E$.

For edges $f, e \in E^{\text{exec}}$, write $f \prec e$ if f strictly precedes e on some directed path in G^{exec}. Since G^{exec} is a DAG, $\prec$ is a strict partial order. For each edge $e \in E^{\text{exec}}$, define

$$\mathcal{H}_e := \sigma\big(Z_e^{\text{in}}, \{Z_f^{\text{in}}, Z_f^{\text{out}} : f \prec e\}\big). \tag{30}$$

Thus, if $e_1, \dots, e_\ell$ occur in this order on a directed path, then $E_{e_h} \in \mathcal{H}_{e_\ell}$ for every $h < \ell$. In particular, outcomes of edges earlier on a directed path ending at e are $\mathcal{H}_e$-measurable. The bound Eq. (6) must hold conditional on this history; an unconditional calibration average does not suffice.

Corollary 20 (Identical stochastic guards)**.** *Under the conditions of Thm. 7, suppose $A_e = a \in (0, 1)$ at every guarded edge and $|\widetilde{F}_j| \le m$ for each ordered barrier $\widetilde{F}_j$, where $m \ge 1$ and $ma < 1$. For any $\delta \in (0, 1)$, define $k_\delta := \frac{\log(1/\delta)}{\log(1/(ma))}$. Then, whenever the induced execution graph contains k such ordered barriers, every integer $k > k_\delta$ guarantees $\mathbb{P}(L_q) < \delta$.*

The homogeneous rule is conditional on the induced execution graph containing the stated ordered barriers; it does not assert that arbitrarily many such separators exist. Its proof is in App. D.8.

C.7 Recipient-exposure tails

Failure probability does not describe how many recipients a failure exposes. We compare a shared guard before fan-out with independent guards after fan-out, keeping each recipient's harmful-pass probability equal to a. For m recipients, let $N_{\text{sh}} = mB$ with $B \sim \text{Bernoulli}(a)$ and $N_{\text{br}} = \sum_{i=1}^m B_i$ with $B_i \overset{\text{iid}}{\sim} \text{Bernoulli}(a)$.

Definition 21 (Recipient exposure count and convex order)**.** For m unauthorized recipients, let $N \in \{0, 1, \dots, m\}$ be the number that receive forbidden content. For integrable random variables X, Y, write $X \le_{\text{cx}} Y$ if $\mathbb{E}[\varphi(X)] \le \mathbb{E}[\varphi(Y)]$ for every convex φ for which the expectations exist. A quantity $\mathbb{E}[\varphi(N)]$ with nondecreasing convex φ, such as $\varphi(n) = n^2$, is a convex penalty on widespread exposure.

For the utility comparison, also assume independent branch useful-survival indicators with common probability u. We require no independence between harmful and useful survival within a branch.

Theorem 22 (Centralization–tail tradeoff)**.** *For the placements above, $m \ge 2$ and $a \in (0, 1)$ imply*

$$\mathbb{E}[N_{\text{sh}}] = \mathbb{E}[N_{\text{br}}] = ma, \tag{31}$$

$$\mathbb{P}(N_{\text{sh}} \ge 1) = a < 1 - (1 - a)^m = \mathbb{P}(N_{\text{br}} \ge 1), \tag{32}$$

$$\mathbb{P}(N_{\text{sh}} = m) = a > a^m = \mathbb{P}(N_{\text{br}} = m), \tag{33}$$

and

$$N_{\mathrm{br}} \leq_{\mathrm{cx}} N_{\mathrm{sh}}. \tag{34}$$

All recipients receive useful content with probability u under shared placement and u^m under independent branch placement.

Thus shared guarding lowers the chance of any exposure, but branch-local guarding lowers every expected convex exposure penalty. In particular,

$$\mathrm{Var}(N_{\mathrm{sh}}) = m^2 a(1-a), \qquad \mathrm{Var}(N_{\mathrm{br}}) = ma(1-a).$$

Convex order does not order every tail probability; Eqs. (32) and (33) rank the two placements in opposite directions. The proof is in App. D.9.

D PROOFS

D.1 EQUIVALENT SIGNATURES

Proof of Thm. 4. We prove the result by structural induction on the recursively defined circuit. Along with the signature formula, we maintain the following auxiliary property for every subcircuit C: $H_C^{\mathrm{out}} \leq H_C^{\mathrm{in}}$ and $Y_C^{\mathrm{out}} \leq Y_C^{\mathrm{in}}$ a.s. Thus a composite circuit, like a primitive guard, cannot create a tracked harmful or useful mode that is absent at its input.

For a primitive guard g, the signature is its calibrated primitive signature by definition, and the no-creation property is exactly (A1).

We first check that no creation is preserved by the circuit constructors. For serial composition $C = C_1 \circ C_2$, let H_0, H_1, H_2 denote the harmful indicators at the input of C_1, the interface between C_1 and C_2, and the output of C_2. By the induction hypothesis, $H_1 \leq H_0$ for C_1 and $H_2 \leq H_1$ for C_2. Hence $H_2 \leq H_1 \leq H_0$. Equivalently, if $H_0 = 0$, then $H_1 = 0$ and therefore $H_2 = 0$. The same argument gives $Y_2 \leq Y_0$ for useful information.

For a parallel block, suppose the tracked harmful mode is absent at the common input. By the induction hypothesis, it is then absent at every branch output, so the harmful branch pattern is $\mathbf{0}$. Assumption (A4) gives $\psi(\mathbf{0}) = 0$, so the switch cannot create harmful information at its output. The same argument using $\phi(\mathbf{0}) = 0$ establishes useful no creation. Thus the auxiliary property is preserved by both constructors.

Serial composition. Again let $C = C_1 \circ C_2$, with harmful indicators H_0, H_1, H_2 at the three corresponding ports. By the inductive signature of C_1 and the definition of the composite survival rate, $A_1 = \mathbb{P}(H_1 = 1 \mid H_0 = 1)$ and $A_C = \mathbb{P}(H_2 = 1 \mid H_0 = 1)$.

The quantity A_2 is the calibrated harmful-survival coordinate of C_2. Assumption (A3) is what allows us to reuse this same rate after placing C_2 behind the upstream circuit C_1. Thus, whenever the intermediate harmful mode is reachable, $\mathbb{P}(H_2 = 1 \mid H_1 = 1,\ \text{upstream } C_1) = A_2$. Since the no-creation property of C_1 gives $H_1 = 1 \Rightarrow H_0 = 1$, we may equivalently write $\mathbb{P}(H_2 = 1 \mid H_1 = 1, H_0 = 1) = A_2$.

Now condition on the two possible values of H_1. If $H_1 = 0$, harmful information has been removed before entering C_2, so the no-creation property of C_2 gives $\mathbb{P}(H_2 = 1 \mid H_1 = 0, H_0 = 1) = 0$ whenever this event is reachable. Therefore, when $A_1 > 0$, the law of total probability gives

$$\begin{aligned} A_C &= \mathbb{P}(H_2 = 1 \mid H_0 = 1) \\ &= \mathbb{P}(H_2 = 1 \mid H_1 = 1, H_0 = 1)\mathbb{P}(H_1 = 1 \mid H_0 = 1) \\ &\quad + \mathbb{P}(H_2 = 1 \mid H_1 = 0, H_0 = 1)\mathbb{P}(H_1 = 0 \mid H_0 = 1) \\ &= A_2 A_1 + 0 \cdot (1 - A_1) = A_1 A_2. \end{aligned} \tag{35}$$

If $A_1 = 0$, then $H_1 = 0$ a.s. conditional on $H_0 = 1$, and no creation implies $H_2 = 0$ a.s. Hence the same identity $A_C = A_1 A_2 = 0$ holds without conditioning on the unreachable event $H_1 = 1$.

The useful-information coordinate is identical. Let Y_0, Y_1, Y_2 denote the useful indicators at the same three ports. By the inductive signature of C_1, $U_1 = \mathbb{P}(Y_1 = 1 \mid Y_0 = 1)$, while $U_C = \mathbb{P}(Y_2 = 1 \mid Y_0 = 1)$. Assumption (A3) ensures that, conditional on the useful mode reaching C_2, its

useful-survival probability remains U_2. If $Y_1 = 0$, the no-creation property gives $Y_2 = 0$. Applying the same two-case law-of-total-probability argument therefore gives $U_C = U_1 U_2$.

Finally, assumption (A5) gives the serial resource coordinates directly: $T_C = T_1 + T_2$ and $K_C = K_1 + K_2$. Combining the four coordinates proves Eq. (2).

Parallel block followed by a switch. Now let $C = \mathrm{FJ}_{\psi,\phi}(C_1, \ldots, C_m)$. Let H_i be the harmful indicator at the output of branch C_i and write $H = (H_1, \ldots, H_m)$. Conditional on harmful information being present at the common input, the inductive signature of branch C_i gives harmful-survival probability A_i. Assumption (A2) says that these harmful branch outcomes are conditionally independent. Hence, for every $h = (h_1, \ldots, h_m) \in \{0,1\}^m$, we have $\mathbb{P}(H = h \mid H_C^{\mathrm{in}} = 1) = \prod_{i=1}^m A_i^{h_i}(1 - A_i)^{1-h_i}$.

We next condition on the realized branch pattern. Assumption (A4) gives the switch's harmful-release probability for each pattern: $\mathbb{P}(H_C^{\mathrm{out}} = 1 \mid H = h) = \psi(h)$. For any nonzero pattern h, no creation in the branches implies that $H = h$ can occur only when $H_C^{\mathrm{in}} = 1$; for $h = \mathbf{0}$, (A4) gives $\psi(\mathbf{0}) = 0$. Thus the same response law applies in the input-conditioned calculation below. Using the law of total probability over all branch patterns,

$$\begin{aligned} A_C &= \mathbb{P}(H_C^{\mathrm{out}} = 1 \mid H_C^{\mathrm{in}} = 1) \\ &= \sum_{h \in \{0,1\}^m} \mathbb{P}(H_C^{\mathrm{out}} = 1 \mid H = h, H_C^{\mathrm{in}} = 1)\mathbb{P}(H = h \mid H_C^{\mathrm{in}} = 1) \\ &= \sum_{h \in \{0,1\}^m} \psi(h) \prod_{i=1}^m A_i^{h_i}(1 - A_i)^{1-h_i}. \end{aligned} \tag{36}$$

Thus the harmful-survival probability of the complete parallel block is the switch response $\psi(h)$ averaged over the distribution of harmful branch-output patterns.

The useful-information coordinate is obtained separately in the same way. Let Y_i be the useful indicator at the output of branch C_i, and write $Y = (Y_1, \ldots, Y_m)$. Conditional on useful information being present at the common input, assumption (A2) gives the following

$$\mathbb{P}(Y = y \mid Y_C^{\mathrm{in}} = 1) = \prod_{i=1}^m U_i^{y_i}(1 - U_i)^{1-y_i}. \tag{37}$$

Assumption (A4) gives the corresponding switch response $\mathbb{P}(Y_C^{\mathrm{out}} = 1 \mid Y = y) = \phi(y)$. Therefore, by the same law-of-total-probability calculation,

$$U_C = \sum_{y \in \{0,1\}^m} \phi(y) \prod_{i=1}^m U_i^{y_i}(1 - U_i)^{1-y_i}. \tag{38}$$

These are separate marginal calculations. Assumption (A2) requires conditional independence across branches for the harmful indicators and, separately, across branches for the useful indicators; it does not require H_i and Y_i to be independent within the same branch.

For resources, assumption (A5) states that every scheduled branch executes. Since the branches run concurrently, the switch acts after the slowest branch finishes, giving $T_C = \max_i T_i + T_J$. Since every branch executes, all branch costs are incurred, together with the switch cost: $K_C = \sum_i K_i + K_J$. This proves Eq. (3).

We have therefore shown that each allowed constructor preserves the no-creation property and determines the parent signature entirely from the signatures of its children and the fixed switch law. Since every finite typed guard circuit is obtained recursively from primitive guards using these constructors, structural induction proves the result. □

D.2 Dual min-plus composition

We first start proving the following lemma:

Lemma 23. *For a finite set J, let $x_j(\lambda) \geq 0$ and suppose $c_j = \lim_{\lambda\to\infty} -\frac{1}{\lambda}\log x_j(\lambda)$ exists in $[0,\infty]$. With $-\log 0 = +\infty$ and $\min \varnothing = +\infty$ we have:*

$$\lim_{\lambda\to\infty} -\frac{1}{\lambda}\log\sum_{j\in J} x_j(\lambda) = \min_{j\in J} c_j.$$

Proof. If $J = \varnothing$, then $\sum_{j\in J} x_j(\lambda) = 0$, so both sides are $+\infty$ by convention. Assume henceforth that $J \neq \varnothing$, let $M = |J|$, and define $x_{\max}(\lambda) := \max_{j\in J} x_j(\lambda)$. Since the largest term is contained in the sum and every term is at most the largest,

$$x_{\max}(\lambda) \leq \sum_{j\in J} x_j(\lambda) \leq M x_{\max}(\lambda).$$

First suppose $c_\star := \min_{j\in J} c_j < \infty$. Then at least one $x_j(\lambda)$, and hence $x_{\max}(\lambda)$, is positive for all sufficiently large λ. Taking logarithms of the preceding inequalities and multiplying by $-1/\lambda < 0$ gives

$$-\frac{1}{\lambda}\log x_{\max}(\lambda) - \frac{\log M}{\lambda} \leq -\frac{1}{\lambda}\log\sum_{j\in J} x_j(\lambda) \leq -\frac{1}{\lambda}\log x_{\max}(\lambda).$$

Because $-\log(\cdot)$ is decreasing,

$$-\frac{1}{\lambda}\log x_{\max}(\lambda) = \min_{j\in J}\left(-\frac{1}{\lambda}\log x_j(\lambda)\right).$$

Since J is finite and each term in this finite minimum converges to c_j, the minimum converges to the minimum of the limits:

$$-\frac{1}{\lambda}\log x_{\max}(\lambda) \longrightarrow \min_{j\in J} c_j = c_\star.$$

Also $(\log M)/\lambda \to 0$. The squeeze theorem therefore gives

$$-\frac{1}{\lambda}\log\sum_{j\in J} x_j(\lambda) \longrightarrow c_\star.$$

It remains to consider the case $c_j = +\infty$ for every $j \in J$. Fix any $L > 0$. For each j, $-\lambda^{-1}\log x_j(\lambda) \to +\infty$, so for all sufficiently large λ, $x_j(\lambda) \leq e^{-\lambda L}$. Since J is finite, one common threshold can be chosen for all j. Hence

$$\sum_{j\in J} x_j(\lambda) \leq M e^{-\lambda L},$$

and therefore

$$-\frac{1}{\lambda}\log\sum_{j\in J} x_j(\lambda) \geq L - \frac{\log M}{\lambda}.$$

Taking $\lambda \to \infty$ gives a lower limit of at least L. Since $L > 0$ was arbitrary, the limit is $+\infty = \min_{j\in J} c_j$. □

Proof of Thm. 13. We use structural induction on the fixed circuit expression. For every subcircuit C, we prove simultaneously that

(i) the privacy exponent $R_C = \lim_{\lambda\to\infty} -\lambda^{-1}\log A_C(\lambda)$ exists and satisfies the claimed composition law;
(ii) the task-failure exponent $S_C = \lim_{\lambda\to\infty} -\lambda^{-1}\log(1 - U_C(\lambda))$ exists and satisfies the claimed composition law;
(iii) The circuit C remains in the rare-error regime: $A_C(\lambda) \to 0$ and $1 - U_C(\lambda) \to 0$.

Base case. If C is a primitive guard, (i)–(iii) are exactly the assumptions in Eq. (16): the limits defining R_C and S_C exist, and both harmful survival and task failure vanish as $\lambda \to \infty$.

Inductive step. Assume (i)–(iii) hold for the immediate child circuits. We prove them for their parent, considering the two constructors separately.

Series. Let $C = C_1 \circ C_2$, and write $f_i(\lambda) := 1 - U_i(\lambda)$ and $f_C(\lambda) := 1 - U_C(\lambda)$.

For (i), Eq. (2) gives $A_C = A_1 A_2$. Hence

$$-\frac{1}{\lambda}\log A_C = -\frac{1}{\lambda}\log A_1 - \frac{1}{\lambda}\log A_2.$$

By the induction hypothesis, the two terms converge to R_1 and R_2, respectively. Therefore

$$R_C = R_1 + R_2,$$

with the same conclusion when one of the rates is $+\infty$.

For (ii), again by Eq. (2),

$$f_C = 1 - U_1 U_2 = f_1 + f_2 - f_1 f_2.$$

Since $0 \le f_1, f_2 \le 1$,

$$\max\{f_1, f_2\} \le f_C \le f_1 + f_2.$$

The logarithmic rate of $\max\{f_1, f_2\}$ is $\min\{S_1, S_2\}$, while Lem. 23 shows that $f_1 + f_2$ has the same rate. Applying $-\lambda^{-1}\log(\cdot)$ to the preceding bounds and using the squeeze theorem gives

$$S_C = \min\{S_1, S_2\}.$$

For (iii), the induction hypothesis gives $A_1, A_2, f_1, f_2 \to 0$. Hence $A_C = A_1 A_2 \to 0$, and

$$0 \le f_C \le f_1 + f_2 \longrightarrow 0.$$

Thus the series parent also remains in the rare-error regime.

Fork–join: privacy. Let $C = \mathrm{FJ}_{\psi,\phi}(C_1, \ldots, C_m)$. For each $I \subseteq [m]$, interpret I as the exact set of branches on which harmful information survives. The corresponding harmful-output pattern is $\mathbf{1}_I$. Under (A2), its probability is

$$p_I^H(\lambda) := \prod_{i \in I} A_i(\lambda) \prod_{i \notin I} (1 - A_i(\lambda)).$$

Conditioning on the realized branch pattern gives

$$A_C(\lambda) = \sum_{I \subseteq [m]} \psi(\mathbf{1}_I)\, p_I^H(\lambda). \tag{39}$$

We first establish (i). By the induction hypothesis, $A_i(\lambda) \to 0$ and its logarithmic rate is R_i. Thus, for $i \notin I$, $1 - A_i(\lambda) \to 1$, so $-\lambda^{-1}\log(1 - A_i(\lambda)) \to 0$. Therefore

$$\lim_{\lambda \to \infty} -\frac{1}{\lambda}\log p_I^H(\lambda) = \sum_{i \in I} R_i.$$

If $\psi(\mathbf{1}_I) = 0$, the pattern contributes nothing to (39). If $\psi(\mathbf{1}_I) > 0$, it is a fixed positive coefficient and therefore has logarithmic rate zero. Applying Lem. 23 to the finite sum gives

$$R_C = \min_{I:\, \psi(\mathbf{1}_I) > 0} \sum_{i \in I} R_i.$$

It remains to show that it suffices to minimize over privacy certificates. Every set I with $\psi(\mathbf{1}_I) > 0$ contains an inclusion-minimal positive subset $L \subseteq I$, since $[m]$ is finite. By definition, $L \in \mathrm{Cert}(\psi)$. Since every $R_i \ge 0$,

$$\sum_{i \in L} R_i \le \sum_{i \in I} R_i.$$

Conversely, every $L \in \mathrm{Cert}(\psi)$ is itself a positive set. Hence

$$R_C = \min_{L \in \mathrm{Cert}(\psi)} \sum_{i \in L} R_i.$$

If there is no positive pattern, then $\psi \equiv 0$, so $A_C = 0$ and $R_C = +\infty$, agreeing with the convention $\min \varnothing = +\infty$.

For the privacy part of (iii), $\psi(\mathbf{0}) = 0$ means that the parent cannot release harmful information unless at least one child has a harmful output. Therefore

$$A_C \le \mathbb{P}(H \neq \mathbf{0}) \le \sum_{i=1}^{m} A_i \longrightarrow 0,$$

where the last step follows from the induction hypothesis and finite m. Thus the fork–join parent also remains in the privacy rare-error regime.

Fork–join: task failure. We now prove (ii) and the remaining part of (iii). Let $f_i(\lambda) = 1 - U_i(\lambda)$, and let $I \subseteq [m]$ denote the exact set of branches that fail the task. The probability of this pattern is

$$p_I^F(\lambda) := \prod_{i \in I} f_i(\lambda) \prod_{i \notin I} (1 - f_i(\lambda)).$$

By the induction hypothesis, $f_i \to 0$ with logarithmic rate S_i, so the same product argument gives

$$\lim_{\lambda \to \infty} -\frac{1}{\lambda} \log p_I^F(\lambda) = \sum_{i \in I} S_i.$$

If precisely the branches in I fail, then the branch-success pattern is $\mathbf{1} - \mathbf{1}_I$. Hence the switch fails the task with probability $1 - \phi(\mathbf{1} - \mathbf{1}_I)$, and Eq. (3) gives

$$f_C(\lambda) = \sum_{I \subseteq [m]} \big[1 - \phi(\mathbf{1} - \mathbf{1}_I)\big] p_I^F(\lambda). \tag{40}$$

As before, a zero coefficient removes a pattern, while a fixed positive coefficient has logarithmic rate zero. Thus Lem. 23 gives

$$S_C = \min_{I:\, \phi(\mathbf{1} - \mathbf{1}_I) < 1} \sum_{i \in I} S_i.$$

Every contributing failed set contains an inclusion-minimal contributing subset. By definition these minimal sets are precisely $\mathrm{Cut}(\phi)$, and since all $S_i \ge 0$,

$$S_C = \min_{I \in \mathrm{Cut}(\phi)} \sum_{i \in I} S_i.$$

This proves (ii).

Finally, for (iii), $\phi(\mathbf{1}) = 1$ implies that the parent cannot fail when every branch succeeds. Hence $\{\text{parent task failure}\} \subseteq \{Y \neq \mathbf{1}\}$. Moreover, $\{Y \neq \mathbf{1}\} = \bigcup_{i=1}^{m} \{Y_i = 0\}$. Therefore, by the union bound,

$$f_C \le \mathbb{P}(Y \neq \mathbf{1}) \le \sum_{i=1}^{m} \mathbb{P}(Y_i = 0) = \sum_{i=1}^{m} f_i \longrightarrow 0,$$

where the last limit follows from the induction hypothesis and finite m. Thus the fork–join parent also satisfies the task part of (iii).

We have now proved (i)–(iii) for each allowed parent constructor, assuming they hold for its immediate children. Together with the primitive base case, structural induction proves the claimed laws for every fixed circuit expression.

For the stated special cases, OR privacy has singleton certificates $\{i\}$, giving exponent $\min_i R_i$, whereas AND privacy has the sole certificate $[m]$, giving $\sum_i R_i$. Task-any and task-all reverse these roles. An at-least-ℓ privacy threshold has all ℓ-subsets as certificates, so its exponent is the sum of the ℓ smallest R_i. Likewise, an at-least-k-of-m task rule fails when at least $m - k + 1$ branches fail, so its exponent is the sum of the $m - k + 1$ smallest S_i. □

D.3 Balanced depth–width design

Proof of Cor. 5. One exposed parallel-any stage of width m has rates (r, ms) by Thm. 13. Composing k such stages in series gives $(R_{k,m}, S_{k,m}) = (kr, ms)$ for fixed integers k, m.

For the continuous optimization, any $k, m > 0$ with $km \le n$ satisfies

$$\min\{kr, ms\} \le \sqrt{(kr)(ms)} = \sqrt{kmrs} \le \sqrt{nrs}.$$

Equality requires $kr = ms$ and $km = n$. Their positive solution is $k^\star = \sqrt{ns/r}$, $m^\star = \sqrt{nr/s}$, which attains the bound. If either coordinate is below one, this remains an algebraic relaxation, not an implementable circuit.

For integer $k \in \{1, \dots, n\}$, the objective is nondecreasing in m, so $m = \lfloor n/k \rfloor$ attains the best value for that k. Checking these n pairs is exact. The optimum may leave some budget unused because of integer constraints. □

D.4 Marginal-only fork–join bounds

Proof of Prop. 14. Work under the relevant input-conditioned probability law. For privacy this means conditioning on the harmful mode being present at the common fork input, and for utility on the useful mode being present. Let E_i denote the corresponding event on branch i, with $\mathbb{P}(E_i) = p_i$. We make no independence assumption among the E_i.

Union bounds. For an any-type output, the relevant event is $\bigcup_i E_i$. Since $E_i \subseteq \bigcup_j E_j$ for every i,

$$\max_i p_i \le \mathbb{P}\left(\bigcup_i E_i\right).$$

The union bound and the fact that probabilities are at most one give

$$\mathbb{P}\left(\bigcup_i E_i\right) \le \min\left\{1, \sum_i p_i\right\}.$$

Hence we have:

$$\max_i p_i \le \mathbb{P}\left(\bigcup_i E_i\right) \le \min\left\{1, \sum_i p_i\right\}.$$

For exposed-any privacy, take $E_i = \{H_i = 1\}$ and $p_i = A_i$, proving Eq. (20). For parallel-any utility, take $E_i = \{Y_i = 1\}$ and $p_i = U_i$, proving Eq. (21).

Intersection bounds. For an all-type output, the relevant event is $\bigcap_i E_i$. Since $\bigcap_i E_i \subseteq E_j$ for every j,

$$\mathbb{P}\left(\bigcap_i E_i\right) \le \min_i p_i.$$

For the lower bound, De Morgan's law and the union bound give

$$\begin{aligned}\mathbb{P}\left(\bigcap_i E_i\right) &= 1 - \mathbb{P}\left(\bigcup_i E_i^c\right) \\ &\ge 1 - \sum_i \mathbb{P}(E_i^c) \\ &= 1 - \sum_i (1 - p_i) = \sum_i p_i - (m-1).\end{aligned}$$

Since probabilities are nonnegative,

$$\max\left\{0, \sum_i p_i - (m-1)\right\} \le \mathbb{P}\left(\bigcap_i E_i\right) \le \min_i p_i.$$

Taking $E_i = \{Y_i = 1\}$ and $p_i = U_i$ proves Eq. (22).

Sharpness. We now show that each endpoint can actually occur, so none of the bounds can be improved using only the marginals $p_1, \dots, p_m$. Work on $[0, 1)$ with uniform measure μ.

For the smallest possible union and largest possible intersection, take the nested events

$$E_i = [0, p_i).$$

Then $\mu(E_i) = p_i$, and

$$\mu\left(\bigcup_i E_i\right) = \max_i p_i, \qquad \mu\left(\bigcap_i E_i\right) = \min_i p_i.$$

Thus the lower union and upper intersection bounds are attained.

For the largest possible union, set $s_0 = 0$ and $s_i = \sum_{j=1}^{i} p_j$, and take

$$E_i = [s_{i-1}, s_i) \pmod 1.$$

Each event has measure p_i. If $\sum_i p_i \le 1$, the intervals are disjoint and their union has measure $\sum_i p_i$; if $\sum_i p_i \ge 1$, wrapping around the unit interval covers all of $[0, 1)$. Therefore

$$\mu\left(\bigcup_i E_i\right) = \min\left\{1, \sum_i p_i\right\},$$

which attains the upper union bound.

Finally, to attain the smallest possible intersection, apply the preceding maximal-union construction to events F_i with $\mu(F_i) = 1 - p_i$, and set $E_i = F_i^c$. Then $\mu(E_i) = p_i$, and

$$\begin{aligned}
\mu\left(\bigcap_i E_i\right) &= 1 - \mu\left(\bigcup_i F_i\right) \\
&= 1 - \min\left\{1, \sum_i (1 - p_i)\right\} \\
&= \max\left\{0, \sum_i p_i - (m - 1)\right\}.
\end{aligned}$$

Thus the lower intersection bound is attained as well. Every endpoint in Eqs. (20) to (22) is therefore achievable by some joint distribution with the prescribed branch marginals, so the bounds are sharp. □

D.5 Monotone substitution

Proof of Thm. 6. We first show that each fixed constructor preserves operational and contract orders, then induct from the replaced slot to the root of $\mathcal{K}$.

Signatures. Series preserves the order because multiplication on $[0, 1]$ and addition of nonnegative resources are nondecreasing in each argument. For a fork, let V_i be independent uniform variables on $[0, 1]$. When $A_i' \le A_i$, define $H_i' = \mathbf{1}\{V_i \le A_i'\}$ and $H_i = \mathbf{1}\{V_i \le A_i\}$. Then $H' \le H$ a.s., with the required independent Bernoulli marginals. Monotonicity of the unchanged table gives

$$A'_{\mathrm{FJ}} = \mathbb{E}[\psi(H')] \le \mathbb{E}[\psi(H)] = A_{\mathrm{FJ}}.$$

Using a separate such coupling for $U_i' \ge U_i$ gives $Y_i' = \mathbf{1}\{V_i \le U_i'\} \ge Y_i = \mathbf{1}\{V_i \le U_i\}$ and hence $U'_{\mathrm{FJ}} \ge U_{\mathrm{FJ}}$. These are comparison couplings, not assumptions about joint privacy and utility in a deployed run. The maps $\max_i T_i + T_J$ and $\sum_i K_i + K_J$ preserve resource order for fixed overheads.

Contracts. Series preserves inclusion through union. At a fork, $\mathsf{B}_{C_i'} \supseteq \mathsf{B}_{C_i}$ implies $(b^q)' \le b^q$. If q belongs to the old parent contract, then

$$0 \le \psi((b^q)') \le \psi(b^q) = 0,$$

so it belongs to the new one by Eq. (9).

Substitution. The claim is immediate when the full expression is the distinguished slot C_i. Otherwise, follow the unique path from that slot to the root of $\mathcal{K}$. At each surrounding series or fork constructor, all companion subcircuits remain fixed, so the preceding monotonicity results preserve $\preceq$. Induction along this path proves Eq. (5). Only weak dominance is required; a strict improvement may become a tie. □

D.6 Budgeted topology synthesis

Proof of Cor. 17. For a circuit C, write $x(C) := (\mathsf{B}_C, \sigma(C))$, and for a record $r = (B, \sigma, \theta)$ write $x(r) := (B, \sigma)$. We first establish, by induction on n, the following two properties:

(i) every record $r \in F_n$ reconstructs a circuit $C \in \mathcal{C}_n$ such that $x(r) = x(C)$;
(ii) for every $C \in \mathcal{C}_n$, there exists a record $r \in F_n$ such that $x(r) \preceq_{\mathrm{cs}} x(C)$.

For $n = 1$, before pruning the candidate set contains exactly the records $(\mathsf{B}_g, \sigma(g), g)$ for $g \in \mathcal{G}$. Hence (i) is immediate. For (ii), if the record of g is retained there is nothing to prove. Otherwise, since the candidate set is finite, following strict dominators until the chain terminates yields a retained record whose state is no worse than $(\mathsf{B}_g, \sigma(g))$; equal states are represented by one retained record.

Now suppose (i)–(ii) hold for all sizes smaller than n. Every $C \in \mathcal{C}_n$ has a root decomposition

$$C = \omega(C_L, C_R), \qquad C_L \in \mathcal{C}_i, \quad C_R \in \mathcal{C}_{n-i},$$

for some $1 \le i < n$ and $\omega \in \Omega$.

To prove (i), let $r \in F_n$. Since Prune only deletes records or merges equal states, r is represented by a candidate generated from some $r_L \in F_i$, $r_R \in F_{n-i}$, and $\omega \in \Omega$. By the induction hypothesis, r_L and r_R reconstruct circuits $C_L \in \mathcal{C}_i$ and $C_R \in \mathcal{C}_{n-i}$ with the same states. The exact contract and signature composition rules therefore give $x(r) = x(\omega(C_L, C_R))$, and $\omega(C_L, C_R) \in \mathcal{C}_n$.

To prove (ii), fix $C = \omega(C_L, C_R) \in \mathcal{C}_n$. By the induction hypothesis there are records $r_L \in F_i$ and $r_R \in F_{n-i}$ satisfying

$$x(r_L) \preceq_{\mathrm{cs}} x(C_L), \qquad x(r_R) \preceq_{\mathrm{cs}} x(C_R).$$

By (i), these records reconstruct circuits C'_L and C'_R with $x(C'_L) = x(r_L)$ and $x(C'_R) = x(r_R)$. Applying Thm. 6 first to one child and then to the other gives

$$x(\omega(C'_L, C'_R)) \preceq_{\mathrm{cs}} x(\omega(C_L, C_R)) = x(C).$$

The algorithm generates the corresponding parent candidate. If that candidate is removed by Prune, finiteness again guarantees a retained record that dominates it. Hence some $r \in F_n$ satisfies $x(r) \preceq_{\mathrm{cs}} x(C)$, proving (ii).

We now prove the equality in the statement. Let $L_n := \{x(r) : r \in F_n\}$ and let R_n denote the set on the right-hand side of Cor. 17.

First we show $L_n \subseteq R_n$: Take $x(r) \in L_n$. By (i), there exists $C \in \mathcal{C}_n$ with $x(r) = x(C)$. Suppose, for contradiction, that $x(r) \notin R_n$. Since $x(r)$ is attainable, there must exist $D \in \mathcal{C}_n$ such that $x(D) \prec_{\mathrm{cs}} x(r)$. By (ii), some $r' \in F_n$ satisfies $x(r') \preceq_{\mathrm{cs}} x(D)$. Therefore, by transitivity, $x(r') \prec_{\mathrm{cs}} x(r)$. Thus one retained state strictly dominates another retained state, which contradicts the definition of Prune. Hence $x(r) \in R_n$.

Second case $R_n \subseteq L_n$: Take $x(C) \in R_n$ for some $C \in \mathcal{C}_n$. By (ii), there exists $r \in F_n$ such that $x(r) \preceq_{\mathrm{cs}} x(C)$. By (i), $x(r)$ is itself attainable by a circuit in $\mathcal{C}_n$. Since $x(C)$ is nondominated among attainable states, the preceding weak dominance cannot be strict. Hence $x(r) = x(C)$, and therefore $x(C) \in L_n$. This proves $L_n = R_n$.

To prove the last step, we have to repeat the similar argument as above which for sake of completeness we provide. Define

$$\mathcal{C}_{\le N} := \bigcup_{n=1}^{N} \mathcal{C}_n, \qquad F_{\le N} := \mathrm{Prune}\left(\bigcup_{n=1}^{N} F_n \right).$$

Similar to before, $L_{\le N} := \{x(r) : r \in F_{\le N}\}$, and let $R_{\le N}$ denote the set of nondominated attainable states $x(C)$ with $C \in \mathcal{C}_{\le N}$.

First, every record in $\bigcup_{n=1}^{N} F_n$ reconstructs, by (i), a circuit in $\mathcal{C}_{\le N}$, and Prune does not create new records. Hence every state in $L_{\le N}$ is attainable.

Moreover, for every $C \in \mathcal{C}_{\le N}$, there is some $n \le N$ with $C \in \mathcal{C}_n$. By (ii), there exists $r \in F_n$ such that $x(r) \preceq_{\mathrm{cs}} x(C)$. If r is removed by the final pruning, finiteness of $\bigcup_{n=1}^{N} F_n$ guarantees a retained record $r' \in F_{\le N}$ satisfying $x(r') \preceq_{\mathrm{cs}} x(r) \preceq_{\mathrm{cs}} x(C)$. Thus every attainable state in $\mathcal{C}_{\le N}$ has a no-worse representative in $F_{\le N}$.

We can now repeat the two-inclusion argument above. If $x(r) \in L_{\le N}$ were strictly dominated by some attainable $x(C)$, the preceding property would give a retained $x(r') \in L_{\le N}$ with $x(r') \preceq_{\mathrm{cs}} x(C) \prec_{\mathrm{cs}} x(r)$, contradicting the definition of Prune. Hence $L_{\le N} \subseteq R_{\le N}$.

Conversely, let $x(C) \in R_{\le N}$. The preceding property gives $r \in F_{\le N}$ with $x(r) \preceq_{\mathrm{cs}} x(C)$. Since $x(r)$ is attainable and $x(C)$ is nondominated, this dominance cannot be strict. Hence $x(r) = x(C)$, so $R_{\le N} \subseteq L_{\le N}$. Therefore $L_{\le N} = R_{\le N}$, which proves the budgeted-frontier statement. □

To solve Eq. (23), retain records with $q \in B$, apply $A \le \bar{A}$, $U \ge \underline{U}$, and $T \le \bar{T}$, and then choose minimum K.

D.7 Ideal placement

Proof of Thm. 18. Write $\lambda = \lambda_{s_q t_q}(\widehat{G}_q)$ and $\delta^+(S) = \{(u,v) \in E(\widehat{G}_q) : u \in S,\ v \notin S\}$.

For any feasible $F \in \mathfrak{F}_q(G)$, let S be the vertices reachable from s_q in $\widehat{G}_q - F$. Feasibility implies $t_q \notin S$. If an edge leaving S were not in F, its head would also be reachable. Thus $\delta^+(S) \subseteq F \subseteq E_g$, and nonnegative costs give

$$\lambda \le c(\delta^+(S)) \le c(F), \qquad \lambda \le \min_{F \in \mathfrak{F}_q(G)} c(F). \tag{41}$$

Conversely, let S attain the minimum directed cut and choose a feasible F_0. The first inequality yields $\lambda \le c(F_0) \le \sum_{e \in E_g} c_e < M_q$. Hence $\delta^+(S)$ contains no edge of capacity M_q, and $F := \delta^+(S) \subseteq E_g$. A forbidden path in $G - F$, together with the auxiliary source and sink edges, would be an s_q–t_q path in $\widehat{G}_q - F$. Such a path must leave S, contradicting deletion of $\delta^+(S)$. Therefore F is feasible and

$$\min_{F' \in \mathfrak{F}_q(G)} c(F') \le c(F) = \lambda. \tag{42}$$

Combining the two directions proves Eq. (29) and constructs an optimal placement. □

D.8 Stochastic privacy barriers

Proof of Thm. 7. For each edge e, set $I_e = \{\mathsf{Rec}_q(Z_e^{\mathrm{in}}) = 1\}$ and $E_e = \{\mathsf{Rec}_q(Z_e^{\mathrm{out}}) = 1\}$. The proof is done in three steps which are stated as follows:

1. Each harmful output has a harmful predecessor. On a guarded edge, $I_e \in \mathcal{H}_e$ and the conditional certificate imply

$$\mathbb{P}(E_e \cap I_e^c) = \mathbb{E}[\mathbf{1}_{I_e^c} \mathbb{P}(E_e \mid \mathcal{H}_e)] \le A_e \mathbb{E}[\mathbf{1}_{I_e^c} \mathbf{1}_{I_e}] = 0. \tag{43}$$

Thus $E_e \subseteq I_e$ a.s.; on an unguarded edge, $E_e = I_e$. If L_q occurs, some incoming edge of an unauthorized recipient occurrence has a harmful output. Trace this edge backward. Unless its tail is in O_q^{exec}, Eq. (7) supplies an earlier incoming harmful edge. The finite DAG prevents an infinite backward chain. Therefore the chain ends in O_q^{exec} and gives a forbidden directed path with E_e true on every edge.

2. Bound each ordered crossing tuple. For every guarded e and every event $D \in \mathcal{H}_e$,

$$\mathbb{P}(D \cap E_e) = \mathbb{E}[\mathbf{1}_D \mathbb{P}(E_e \mid \mathcal{H}_e)] \le A_e \mathbb{P}(D \cap I_e) \le A_e \mathbb{P}(D). \tag{44}$$

Put $s_j = \sum_{e \in \widetilde{F}_j} A_e$ and retain only $\mathcal{I} = \{j : s_j < 1\}$. The other barriers contribute the factor one to the claimed bound. If $\mathcal{I} = \varnothing$, the claim is $\mathbb{P}(L_q) \le 1$ which is trivially correct. Otherwise write $\mathcal{I} = \{j_1 < \cdots < j_r\}$.

Let $\mathcal{T}$ be the tuples $(e_1, \dots, e_r) \in \prod_{\ell=1}^{r} \widetilde{F}_{j_\ell}$ that occur in that order along some forbidden directed path. For a fixed tuple set D_0 equal to the whole sample space and $D_\ell = \bigcap_{h=1}^{\ell} E_{e_h}$. For $h < \ell$, edge e_h precedes e_ℓ on that path, so $D_{\ell-1} \in \mathcal{H}_{e_\ell}$ by the history definition. Equation (44) gives $\mathbb{P}(D_\ell) \le A_{e_\ell} \mathbb{P}(D_{\ell-1})$. Applying this inductively yields the following bound

$$\mathbb{P}\left(\bigcap_{\ell=1}^{r} E_{e_\ell}\right) \le \prod_{\ell=1}^{r} A_{e_\ell}. \tag{45}$$

3. Sum over possible crossings. By Step 1 and orderedness, every leakage realization contains a tuple from $\mathcal{T}$. The union bound and nonnegative terms give

$$\begin{aligned}\mathbb{P}(L_q) &\le \sum_{(e_1,\dots,e_r)\in\mathcal{T}} \mathbb{P}\left(\bigcap_{\ell=1}^{r} E_{e_\ell}\right) \\ &\le \sum_{(e_1,\dots,e_r)\in\prod_{\ell=1}^{r} \widetilde{F}_{j_\ell}} \prod_{\ell=1}^{r} A_{e_\ell} = \prod_{\ell=1}^{r} s_{j_\ell} = \prod_{j=1}^{k} \alpha(\widetilde{F}_j).\end{aligned}$$

This also covers an empty selected barrier, when the product is zero. Finally, $\mathcal{R}(\widetilde{F}_j) = -\log \alpha(\widetilde{F}_j)$ gives the exponential form stated in Thm. 7. No independence between edges or barriers is used. □

Single-barrier placement on a fixed execution graph. Fix G^{exec} and one conditional bound A_e per guardable execution edge. For one barrier the certified risk is $\alpha(\widetilde{F}) = \min\{1, \sum_{e\in\widetilde{F}} A_e\}$. The clipping function $x \mapsto \min\{1, x\}$ is nondecreasing, so minimizing $\sum_{e\in\widetilde{F}} A_e$ minimizes $\alpha(\widetilde{F})$. Applying Thm. 18 to G^{exec} with weights A_e gives a risk-weighted directed minimum cut. If G itself is a finite single-use DAG, then $G^{\text{exec}} = G$ and this is directly a placement on the workflow. For a cyclic workflow, several execution edges may map through π to the same static deployment edge, so optimizing one execution cut is not by itself a globally optimal static placement on G. If every feasible sum is at least one, all certificates are trivial; clipping may therefore create additional minimizers. The optimization is of the bound, not the unknown exact leakage, a cost-constrained design, or a globally optimal sequence of barriers.

Proof of Cor. 20. Since $A_e = a$ and $|\widetilde{F}_j| \le m$, $\alpha(\widetilde{F}_j) = \min\{1, a|\widetilde{F}_j|\} \le ma$. The barrier theorem implies $\mathbb{P}(L_q) \le (ma)^k$. For $0 < ma < 1$,

$$k > \frac{\log(1/\delta)}{\log(1/(ma))} \implies k\log(1/(ma)) > \log(1/\delta) \implies (ma)^k < \delta.$$

Thus the smallest integer certified by this strict threshold is $\lfloor k_\delta \rfloor + 1$, provided that many ordered barriers exist. □

D.9 Centralized versus branch-local exposure

Proof of Thm. 22. For $N_{\text{sh}} = mB$, $\mathbb{E}[N_{\text{sh}}] = ma$ and $\mathbb{P}(N_{\text{sh}} \ge 1) = \mathbb{P}(N_{\text{sh}} = m) = a$. Independence of the branch indicators gives

$$\mathbb{E}[N_{\text{br}}] = ma, \qquad \mathbb{P}(N_{\text{br}} \ge 1) = 1 - (1-a)^m, \qquad \mathbb{P}(N_{\text{br}} = m) = a^m.$$

For $m \ge 2$ and $0 < a < 1$, $(1-a)^m < 1 - a$ and $a^m < a$, proving Eqs. (31) to (33).

For any convex $\varphi : [0, m] \to \mathbb{R}$, the chord inequality gives

$$\varphi(x) \le (1 - x/m)\varphi(0) + (x/m)\varphi(m) \qquad (0 \le x \le m).$$

Substituting $x = N_{\text{br}}$ and taking expectations yields

$$\mathbb{E}[\varphi(N_{\text{br}})] \le (1-a)\varphi(0) + a\varphi(m) = \mathbb{E}[\varphi(N_{\text{sh}})],$$

which is Eq. (34). Moreover, $\text{Var}(mB) = m^2 a(1-a)$ and independence gives $\text{Var}(\sum_i B_i) = ma(1-a)$.

For utility, a shared guard's single successful output reaches all recipients, with probability u. Under the separate useful-indicator independence assumption, all m branch successes occur with probability u^m. □

E Experiment Details and Additional Results

This appendix provides details for the experiments in Sec. 6. We describe the datasets (Sec. E.1), guards (Sec. E.2), models (Sec. E.3), communication topologies (Sec. E.4), metrics (Sec. E.5), and scoring protocol (Sec. E.6), followed by per-model composition results (Sec. E.7), results for the editing guard (Sec. E.8), full placement results (Sec. E.9), the recurring cut (Sec. E.10), and latency (Sec. E.11).

E.1 Datasets

For each benchmark, we convert every case into a common record containing the task, the private input, the protected items, the task requirements, and provenance information. We insert the protected items directly into the private input, so that every case contains a leakage opportunity. This setting measures whether protected information survives a guard or a workflow, which is a separate question from how often such information arises naturally. The private input is fixed before any guard runs, and every condition for a case uses the same input. Leakage and utility are measured on the final released output, never on the internal messages between agents.

AgentCIBench (ACIB) (Goel and Gurevych, 2026). ACIB provides computer-use scenarios in which an agent operates over a synthetic set of applications under an explicit rule about what may be shared in each context. The task is the requested action, the private input is the synthetic application state, and the scenario labels which items are protected and which are required. Since the items can be matched exactly, ACIB is scored automatically. Several cases can be variants of the same underlying scenario; we cluster by scenario so that these variants are not counted as independent.

AgentSocialBench (ASB) (Wang and Jiang, 2026). ASB provides multi-agent social scenarios in which the final message delivered to the user is scored for both privacy and task completion. We use its cross-domain scenarios. The protected items are the details each agent is instructed not to share, and task success is defined by each scenario's completion criteria. Only the released final message is scored.

PAC-Bench (Park et al., 2026). PAC-Bench is a multi-turn benchmark in which two agents, each holding a private organizational context, negotiate a shared deliverable under their respective privacy policies. Since our experiments concern the released deliverable rather than the negotiation, we run the negotiation once and use its final shared document as the private input; the input never contains the multi-turn conversation. All depths k reuse this document. The protected items are the policy-sensitive terms that each agent must not reveal, and the task requirements are the requirements of the shared goal. We also record the benchmark's own task, privacy, and joint accuracy scores, adapted to the final document; this adaptation does not reproduce the benchmark's per-turn scoring.

E.2 Guards

Each guard processes a single message and receives only the public task and the incoming message. No guard receives the list of protected items, which is held only by the scorer. Instead, the privacy instruction names broad categories of sensitive information; a guard told in advance which terms to remove would measure instruction-following rather than detection. Both guards use the same system prompt, which frames the guard as a downstream relay stage and includes the public task, followed by a guard-specific instruction. The main text uses the censoring guard. We evaluate the editing guard in Experiment 1 on ACIB and ASB only (Sec. E.8).

Censoring guard. The censoring guard returns the rewritten message directly. Its privacy instruction is:

> Reproduce the upstream message unchanged except for specific medical, financial, or otherwise sensitive personal details, which you must delete entirely. Do not summarize, paraphrase, generalize, or replace them with placeholders. Do not add any content of your own.

Since the output is model-written text, the censoring guard can both remove and unintentionally introduce content.

Editing guard. The editing guard asks the model to return the sensitive spans in the message as structured output. Deterministic code then replaces each span with a placeholder generated from a closed set of entity types (e.g., `[NAME1]`). The released message therefore contains only the original text and these placeholders; model-written text never reaches the output, which makes the output auditable.

No-guard baseline. The no-guard baseline forwards each message unchanged. It corresponds to k=0 in Experiment 1 and m=0 in Experiment 2.

E.3 Models

We implement each guard with six hosted LMs: GPT-5-mini and GPT-5.4 (OpenAI), Claude Haiku 4.5 and Claude Sonnet 4.6 (Anthropic), and Gemini 3.1 Flash-Lite and Gemini 3.5 Flash (Google). We vary only the guard model and hold all other agents fixed, so that differences in outcome are attributable to the guard. In Experiment 1, the guards act on a fixed private input and no other agents are involved. In Experiment 2, all non-guard agents use GPT-5-mini. Before the main runs, we checked that each model returns well-formed output, supports the required context length, and is compatible with the evaluator. Each run records the exact model version, the decoding and reasoning settings, and the context limit. Since some reasoning models fix their temperature and not every provider exposes a random seed, we do not impose a common temperature or seed across providers; we record the settings each provider accepts. Experiments were performed on an Apple MacBook Pro (M4 Max, 64 GB of memory); all LMs were accessed through their providers' hosted APIs.

E.4 Communication topologies and placements

Fixed topologies. Both topologies are acyclic and implement a three-stage workflow of propose, review, and resolve between a private source and an external recipient (Fig. 8). The agents are task specialists: a planner, candidate and alternative authors, evidence and constraint checkers, a critic, a reconciler, a synthesizer, and a release formatter. No agent, including the guard, receives the list of protected items.

The Early-Cut topology (EC) implements the three stages directly: the first stage produces candidate and alternative proposals, the second reviews them in parallel, and the third reconciles them and releases the result. Its minimum cut is the unique pair of edges between the propose and review stages, which is strictly cheaper than guarding every edge leaving the source or every edge entering the recipient. The Late-Cut topology (LC) follows the same workflow, but its propose–review boundary is multiply connected, so its unique minimum cut lies one stage later, between the review and resolve stages. Both topologies have 17 guardable edges. Since the minimum cuts lie at different stages, a placement effect that holds on both topologies is not an artifact of one particular structure.

Placements. For each topology and repetition, we draw a uniformly random ordering of the 17 guardable edges and guard the first m edges, for $m \in \{0, \dots, 17\}$. Each prefix is therefore a uniformly random set of m edges, and a single ordering traces the full curve from no guards to every edge guarded. We do not redraw an ordering whose prefix happens to separate the source from the recipient; such a placement counts toward the random policy. We compare the minimum cut against the random placement at m=2. Each repetition draws a new ordering and new guard calls, so the variation reflects both the placement and the stochasticity of the guard. With C cases, B repetitions, and M_g guardable edges in topology g, the number of placement runs is $C \cdot B \cdot \sum_g (M_g + 2)$, consisting of $M_g + 1$ random budgets and the minimum cut. We use B=20 repetitions and report the two topologies separately.

E.5 Metrics

For every dataset, case, configuration, and repetition, we record:

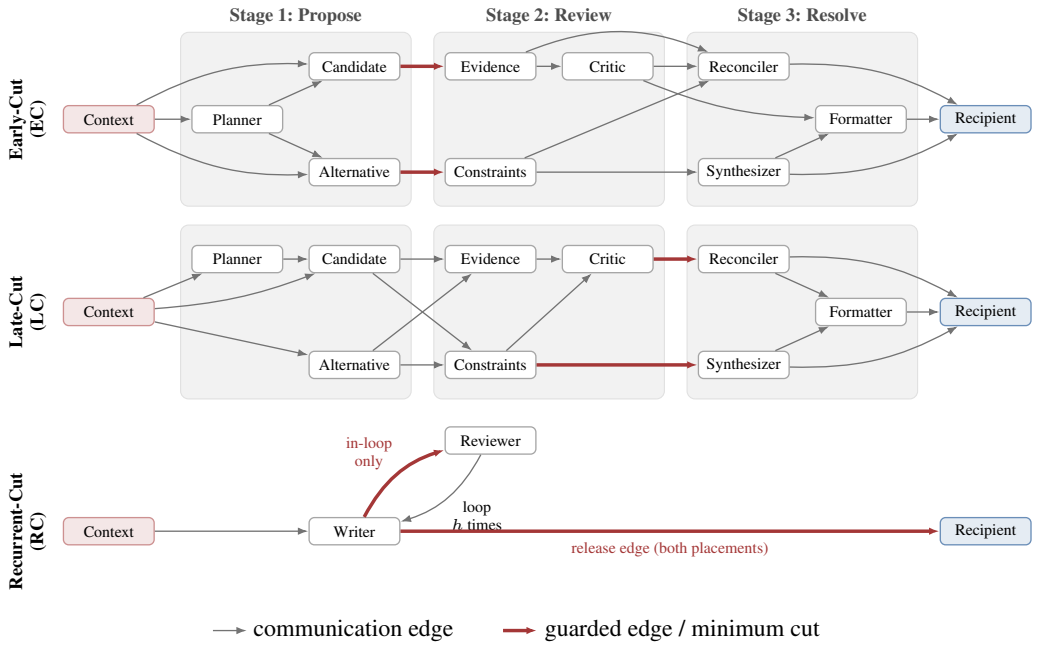

Figure 8: The communication topologies of Experiment 2. Information flows from the private source (left) to the external recipient (right). **EC and LC** are acyclic, and each agent belongs to one of the three workflow stages (shaded), aligned across the two rows. Each has 17 guardable edges and a unique two-edge minimum cut (red), which lies between the propose and review stages in EC and between the review and resolve stages in LC. The names refer only to the location of the cut. **RC** is the writer–reviewer loop, which runs h times before the writer's draft is released. The in-loop placement guards the writer's outgoing edge, which the draft crosses once per iteration; the exit-only placement guards only the release edge.

- **Leakage:** the fraction of protected items that appear in the released output (our primary privacy metric), and $\mathrm{AnyLeak}$, which indicates whether any protected item appears. A case with no protected items has zero leakage.
- **Task Success (TS):** whether the released output satisfies every task requirement. Each dataset also provides a partial score (e.g., the fraction of requirements satisfied), which we record but do not report.
- **Safe Task Success (STS):** whether $\mathrm{TS} = 1$ and $\mathrm{AnyLeak} = 0$.
- **Latency:** the measured wall-clock time and, for parallel structures, the ideal time along the longest branch.

Outputs on which the evaluator fails are excluded and counted separately, rather than recorded as safe or failed. We compare conditions with case-paired one-sided t-tests and Holm-Bonferroni correction (Sec. 6.1); bands in the figures are 95% bootstrap intervals. We do not pool results across datasets, since the benchmarks differ in what a case represents and in how their labels are distributed. In Experiment 1, the main comparison is leakage in series versus parallel at $k{=}5$. In Experiment 2, the main comparison is the difference in leakage between the minimum cut and a random placement of two edges, paired by case, topology, and repetition.

E.6 Scoring protocol

We separate generation from scoring. Every intermediate and final output is saved before scoring begins, and scores are written to separate files without modifying the saved outputs. ACIB is scored by exact, case-insensitive matching of the labeled items. ASB and PAC-Bench are scored by an LM judge (GPT-5.1). For each released output, the judge decides whether each protected item is disclosed, in any wording, and whether each task requirement is satisfied. The judge prompt does not

reveal which guard, model, or structure produced the output. The judge model and prompt were fixed before the main runs.

E.7 PER-MODEL COMPOSITION CURVES

Figure 4 shows the composition curves for GPT-5-mini. Table 4 gives the full-detail leakage contrast at k=5 for every guard model, with case counts and raw p-values. Figures 9 and 10 show the same curves for every guard model, with one row per model and one column per dataset. Each panel has its own vertical scale, since leakage levels differ substantially across datasets.

Guard model	n	Series	Parallel	p
ACIB				
GPT-5-mini	351	0.060	**0.171**	$\mathbf{1.16 \times 10^{-21}}$
GPT-5.4	351	0.114	**0.288**	$\mathbf{1.21 \times 10^{-25}}$
Haiku 4.5	350	0.033	**0.209**	$\mathbf{2.07 \times 10^{-21}}$
Sonnet 4.6	350	0.062	**0.076**	$\mathbf{1.51 \times 10^{-4}}$
Gem. 3.1 FL	351	0.039	**0.101**	$\mathbf{3.02 \times 10^{-14}}$
Gem. 3.5 F	351	0.096	**0.165**	$\mathbf{3.46 \times 10^{-14}}$
ASB				
GPT-5-mini	319	0.051	**0.166**	$\mathbf{6.38 \times 10^{-14}}$
GPT-5.4	321	0.072	**0.120**	$\mathbf{4.99 \times 10^{-6}}$
Haiku 4.5	321	0.052	**0.085**	$\mathbf{1.27 \times 10^{-4}}$
Sonnet 4.6	319	0.055	**0.072**	**0.007**
Gem. 3.1 FL	321	0.056	**0.083**	$\mathbf{3.39 \times 10^{-4}}$
Gem. 3.5 F	320	0.062	**0.171**	$\mathbf{6.01 \times 10^{-12}}$
PAC-Bench				
GPT-5-mini	4428	0.702	**0.880**	$\mathbf{9.14 \times 10^{-236}}$
GPT-5.4	4399	0.681	**0.835**	$\mathbf{1.96 \times 10^{-203}}$
Haiku 4.5	4307	0.453	**0.587**	$\mathbf{2.40 \times 10^{-144}}$
Sonnet 4.6	3620	0.490	**0.551**	$\mathbf{4.31 \times 10^{-41}}$
Gem. 3.1 FL	4424	0.573	**0.719**	$\mathbf{1.92 \times 10^{-179}}$
Gem. 3.5 F	4364	0.834	**0.932**	$\mathbf{1.52 \times 10^{-125}}$

Table 4: Full-detail leakage contrasts at k=5 for the censoring guard (companion to Tab. 2). n is the number of paired cases; p is the raw p-value underlying the significance markings in Tab. 2 (Sec. 6.1).

E.8 EDITING GUARD

We repeat Experiment 1 with the editing guard (Sec. E.2) on ACIB and ASB; PAC-Bench was not run for this guard. Table 5 gives the full-detail leakage contrast at k=5 for every guard model, and Figs. 11 and 12 show the curves for every guard model, with the same layout as Fig. 9. The editing guard follows the same pattern as the censoring guard for every model: series depth reduces leakage and parallel width increases it.

E.9 FULL PLACEMENT RESULTS

Tables 6 and 7 give the raw p-values behind Tab. 3 for the censoring guard on ACIB.

E.10 RECURRING CUT

This test targets the part of Thm. 7 that acyclic topologies cannot reach: a guard on a workflow edge mediates every execution of that edge, so one placement on a recurring edge yields several ordered barriers.

Design. A writer agent A receives a case's task and private input and drafts a recipient-ready response; a reviewer agent B returns feedback without rewriting the draft; after h iterations, A's current draft is released to an external recipient (Fig. 8). Both agents use GPT-5-mini and receive no privacy instruction. The private input is supplied only at iteration 0, so every path to the recipient passes through A's outgoing edge. The *in-loop* arm guards that edge, so the guard mediates all h+1

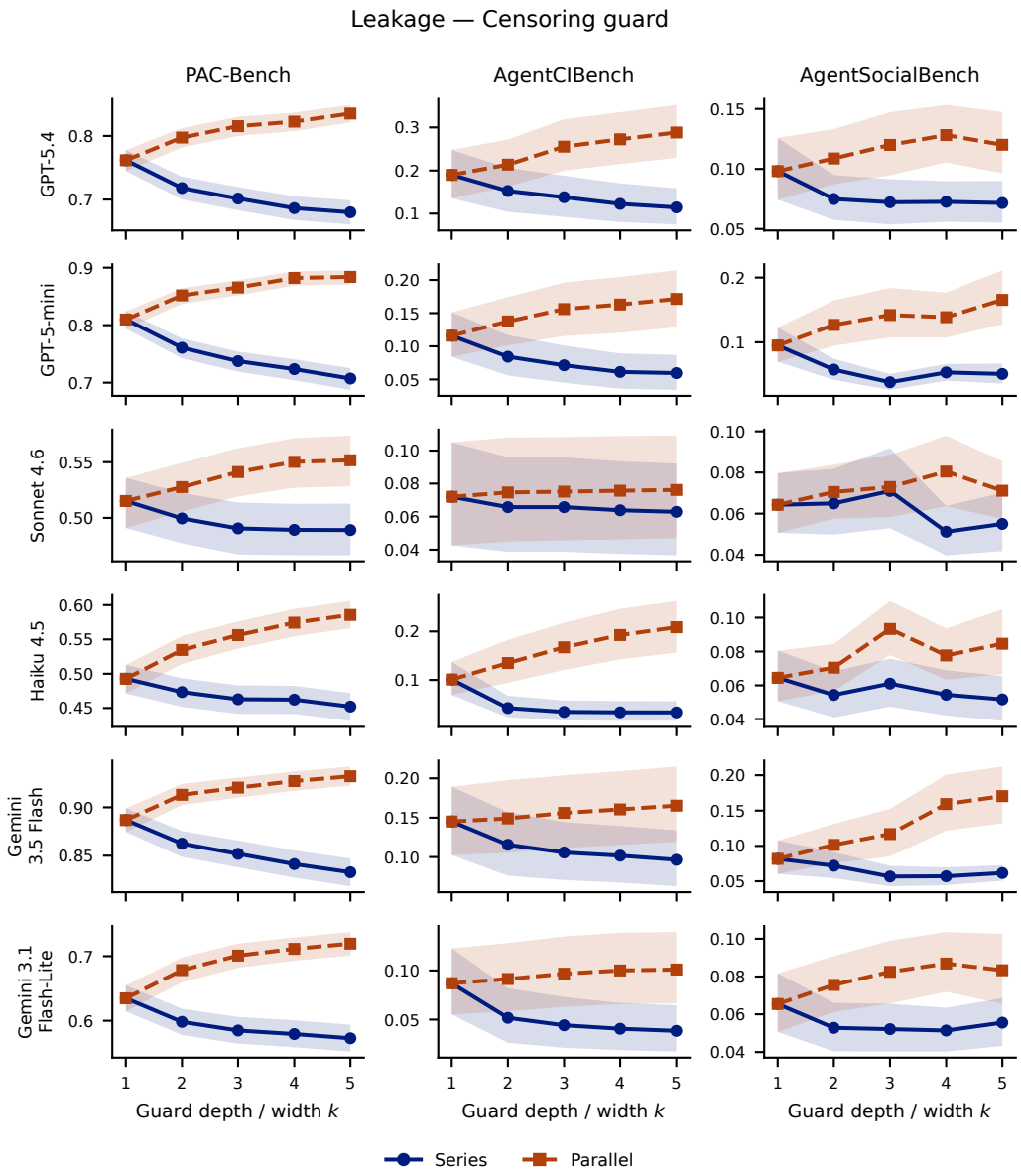

Figure 9: Leakage against the number of guards k for the censoring guard, with one row per guard model and one column per dataset. Bands are 95% bootstrap intervals clustered by scenario.

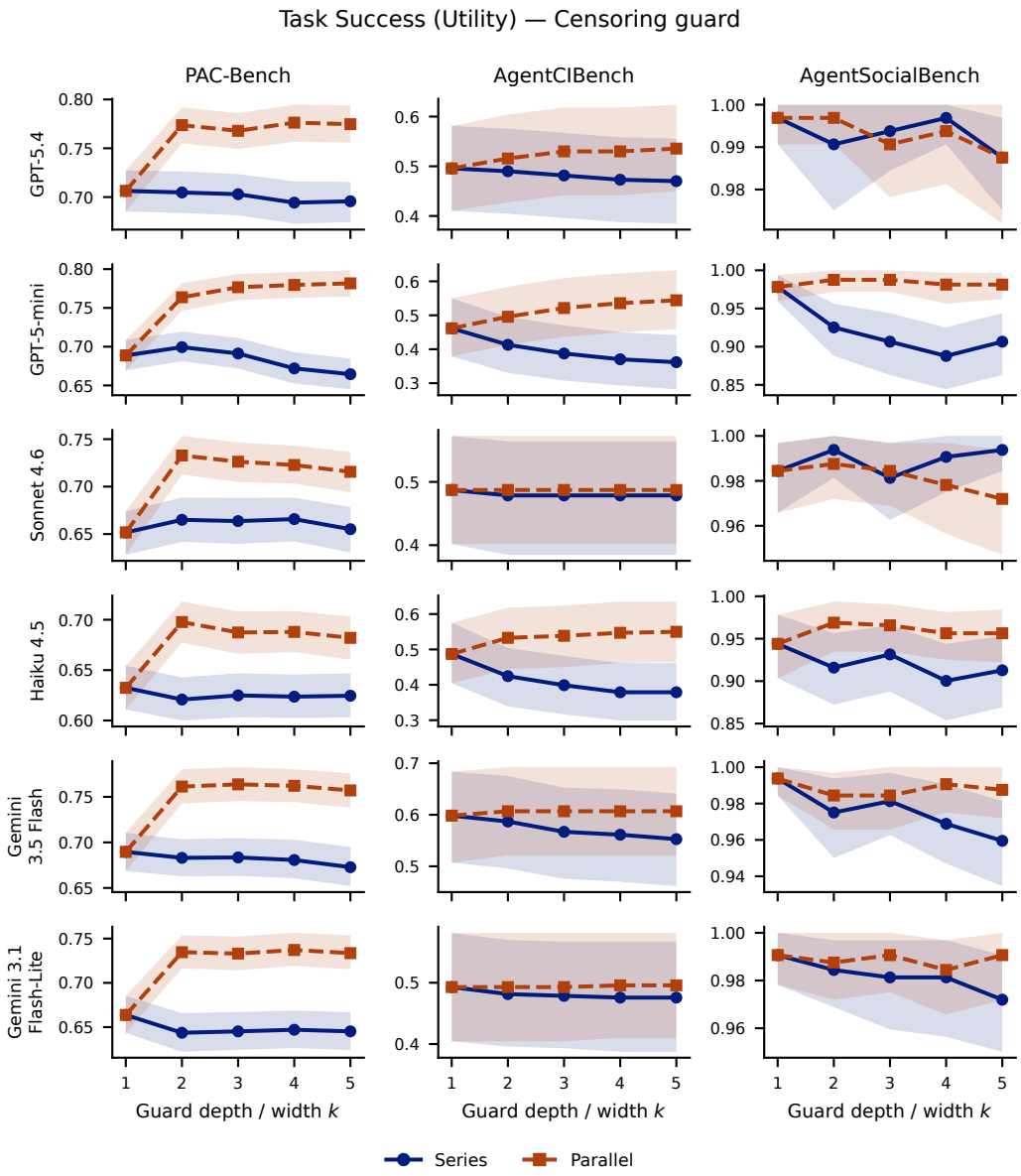

Figure 10: Task Success against the number of guards k for the censoring guard. Same layout and intervals as Fig. 9.

Guard model	n	Series	Parallel	p
ACIB				
GPT-5-mini	349	0.146	**0.299**	$\mathbf{1.57 \times 10^{-21}}$
GPT-5.4	351	0.188	**0.318**	$\mathbf{2.01 \times 10^{-19}}$
Haiku 4.5	351	0.280	**0.404**	$\mathbf{1.38 \times 10^{-20}}$
Sonnet 4.6	351	0.188	**0.230**	$\mathbf{4.55 \times 10^{-9}}$
Gem. 3.1 FL	351	0.228	**0.338**	$\mathbf{3.90 \times 10^{-16}}$
Gem. 3.5 F	351	0.236	**0.372**	$\mathbf{9.19 \times 10^{-18}}$
ASB				
GPT-5-mini	321	0.556	**0.747**	$\mathbf{1.76 \times 10^{-11}}$
GPT-5.4	315	0.529	**0.769**	$\mathbf{1.01 \times 10^{-17}}$
Haiku 4.5	320	0.755	**0.897**	$\mathbf{3.59 \times 10^{-9}}$
Sonnet 4.6	318	0.725	**0.825**	$\mathbf{3.90 \times 10^{-5}}$
Gem. 3.1 FL	321	0.812	**0.893**	$\mathbf{1.39 \times 10^{-4}}$
Gem. 3.5 F	318	0.713	**0.851**	$\mathbf{6.86 \times 10^{-8}}$

Table 5: Full-detail leakage contrasts at k=5 for the editing guard, all 6 guard models. Columns as in Tab. 4. PAC-Bench was not run for the editing guard.

Guard model	Leak. (min cut)	Leak. (random m=2)	p
EC			
GPT-5-mini	**0.164**	0.676	**0.008**
GPT-5.4	**0.226**	0.673	**0.008**
Haiku 4.5	**0.205**	0.655	**0.009**
Sonnet 4.6	**0.164**	0.654	**0.007**
Gem. 3.1 FL	**0.174**	0.687	**0.006**
Gem. 3.5 F	**0.161**	0.702	**0.007**
LC			
GPT-5-mini	**0.163**	0.613	**0.007**
GPT-5.4	**0.238**	0.593	**0.006**
Haiku 4.5	**0.207**	0.667	**0.006**
Sonnet 4.6	**0.181**	0.613	**0.008**
Gem. 3.1 FL	**0.189**	0.648	**0.007**
Gem. 3.5 F	**0.159**	0.607	**0.008**

Table 6: Full-detail leakage levels for the minimum cut versus two random edges (m=2), all 6 guard models, censoring guard. p is the raw p-value underlying the significance markings in Tab. 3 (Sec. 6.1).

Guard model	STS (min cut)	STS (random m=2)	p
EC			
GPT-5-mini	**0.577**	0.114	**0.009**
GPT-5.4	**0.550**	0.107	**0.008**
Haiku 4.5	**0.586**	0.114	**0.011**
Sonnet 4.6	**0.643**	0.150	**0.006**
Gem. 3.1 FL	**0.643**	0.107	**0.006**
Gem. 3.5 F	**0.686**	0.079	**0.004**
LC			
GPT-5-mini	**0.600**	0.224	**0.022**
GPT-5.4	**0.457**	0.179	**0.016**
Haiku 4.5	**0.379**	0.114	**0.032**
Sonnet 4.6	**0.550**	0.164	**0.010**
Gem. 3.1 FL	**0.543**	0.150	**0.008**
Gem. 3.5 F	**0.571**	0.150	**0.012**

Table 7: Full-detail STS levels for the minimum cut versus two random edges (m=2), all 6 guard models, censoring guard. Columns as above.

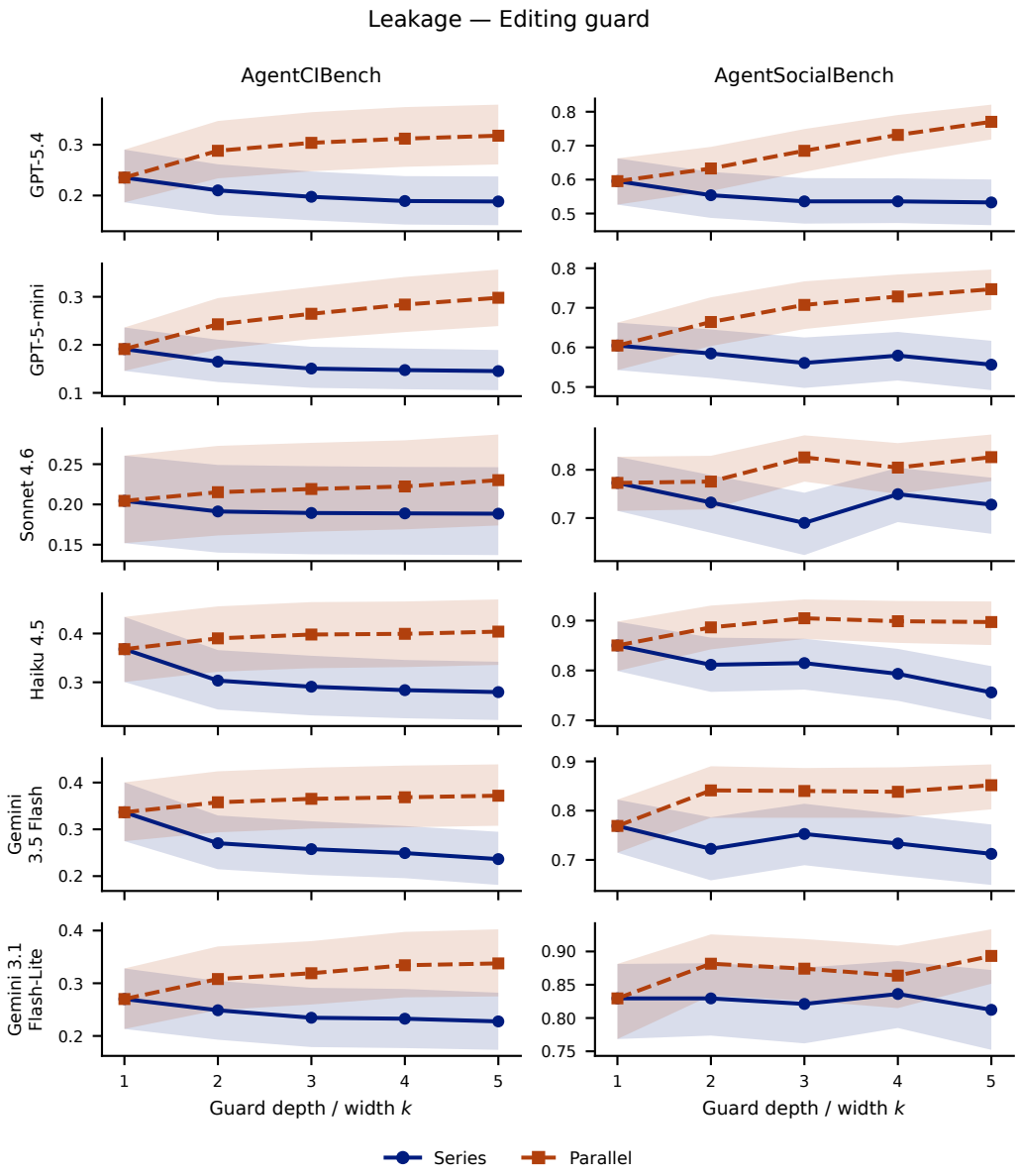

Figure 11: Leakage against the number of guards k for the editing guard. Same layout and intervals as Fig. 9.

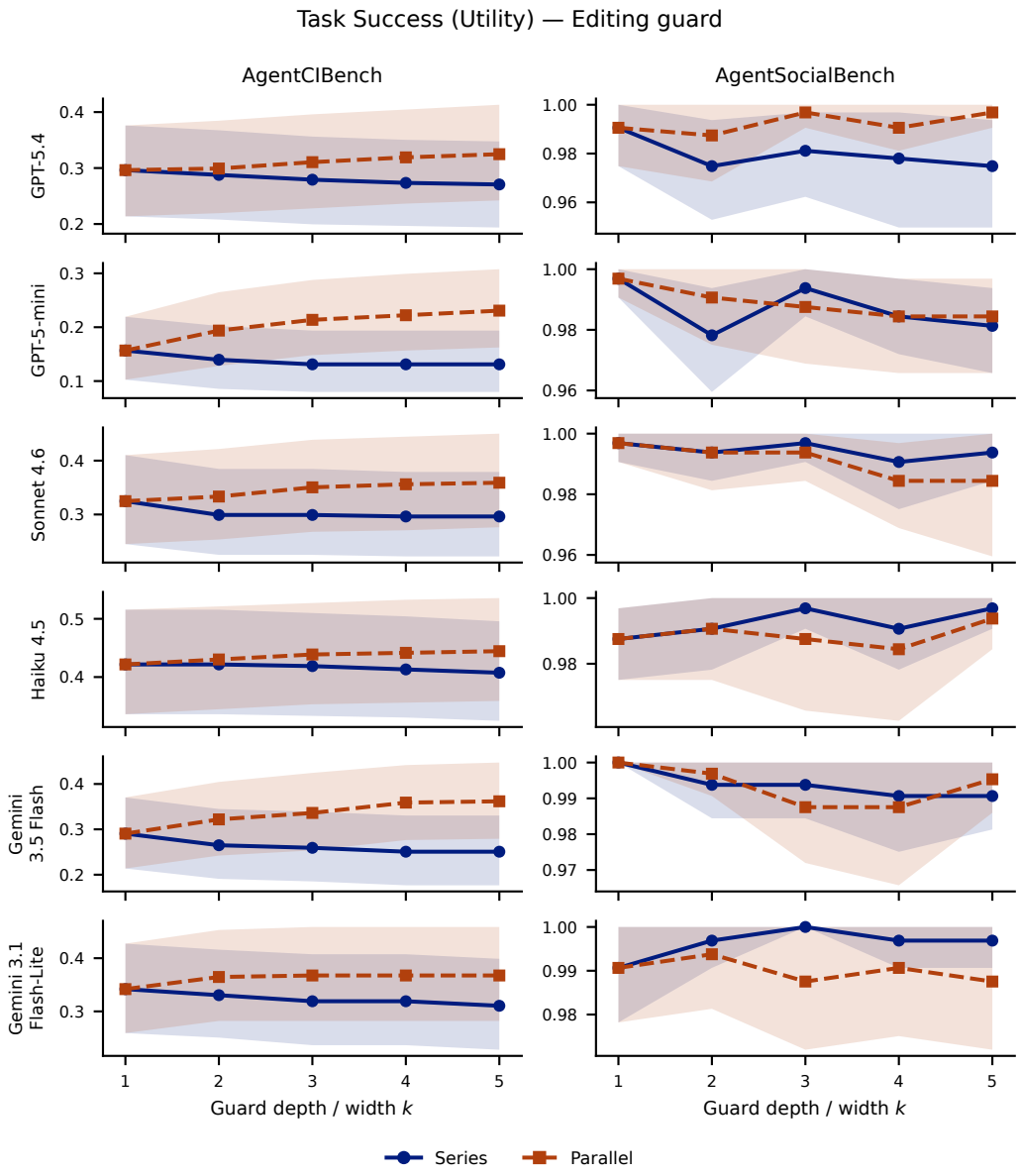

Figure 12: Task Success against the number of guards k for the editing guard. Same layout and intervals as Fig. 9.

of A's messages; the *exit-only* arm applies the same guard once, to the released draft of the unguarded run; the *no-guard* arm forwards every message unchanged. The two guarded arms coincide at $h{=}0$.

Protocol. We use 22 ACIB cases, 20 replicates, and $h \in \{0, \ldots, 5\}$, with the private input fixed across arms and horizons; tests pair by case. Table 9 gives the case-paired contrasts and raw p-values behind Tab. 8.

	Leakage (↓)			STS (↑)		
h	No guard	Exit only	In-loop	No guard	Exit only	In-loop
0	.604	.070	.071	.057	.330	.323
1	.579	.160	.076	.032	.291	.320
2	.567	.191	.076	.027	.239	.302
3	.551	.215	.077	.027	.211	.291
4	.534	.201	.078	.027	.223	.270
5	.525	.224	.079	.027	.198	.259

Table 8: Leakage (↓) and STS (↑) against the number of writer–reviewer iterations h, for the censoring guard on ACIB with GPT-5-mini. *In-loop* guards the writer's outgoing edge; *exit only* guards the release edge alone.

	Leakage: exit − in-loop		STS: in-loop − exit	
h	Δ	p	Δ	p
1	**+.084**	**.0007**	+.030	.2045
2	**+.115**	**.0013**	+.064	.0455
3	**+.138**	**.0008**	+.080	.0156
4	**+.123**	**.0008**	+.048	.0872
5	**+.145**	**.0020**	+.061	.0543

Table 9: Case-paired differences and raw p-values behind Tab. 8 ($n{=}22$ cases). Leakage is exit only minus in-loop, and STS is in-loop minus exit only, so positive values favour the in-loop arm. Significance is marked as in Sec. 6.1, with Holm-Bonferroni correction across horizons. $h{=}0$ is omitted because the two arms are identical there.

E.11 Latency

Figures 13 and 14 show latency against k for every guard model and dataset, for the censoring and editing guards. We run the $k{=}5$ circuit once per case and evaluate smaller k as a prefix of it (Sec. 6.2), so only $k{=}5$ has a directly measured parallel wall-clock time; at every k we instead plot an *estimated critical-path* latency built from each guard call's own measured latency, so both structures are comparable throughout. In series this is the sum of the k stage latencies, since the stages genuinely run one after another at every depth. In parallel it is the maximum of the k branch latencies; i.e., how long the slowest of those k branches would take running concurrently on their own. The parallel curve is therefore a real measurement at $k{=}5$, and all measurements $k{<}5$ are estimated.

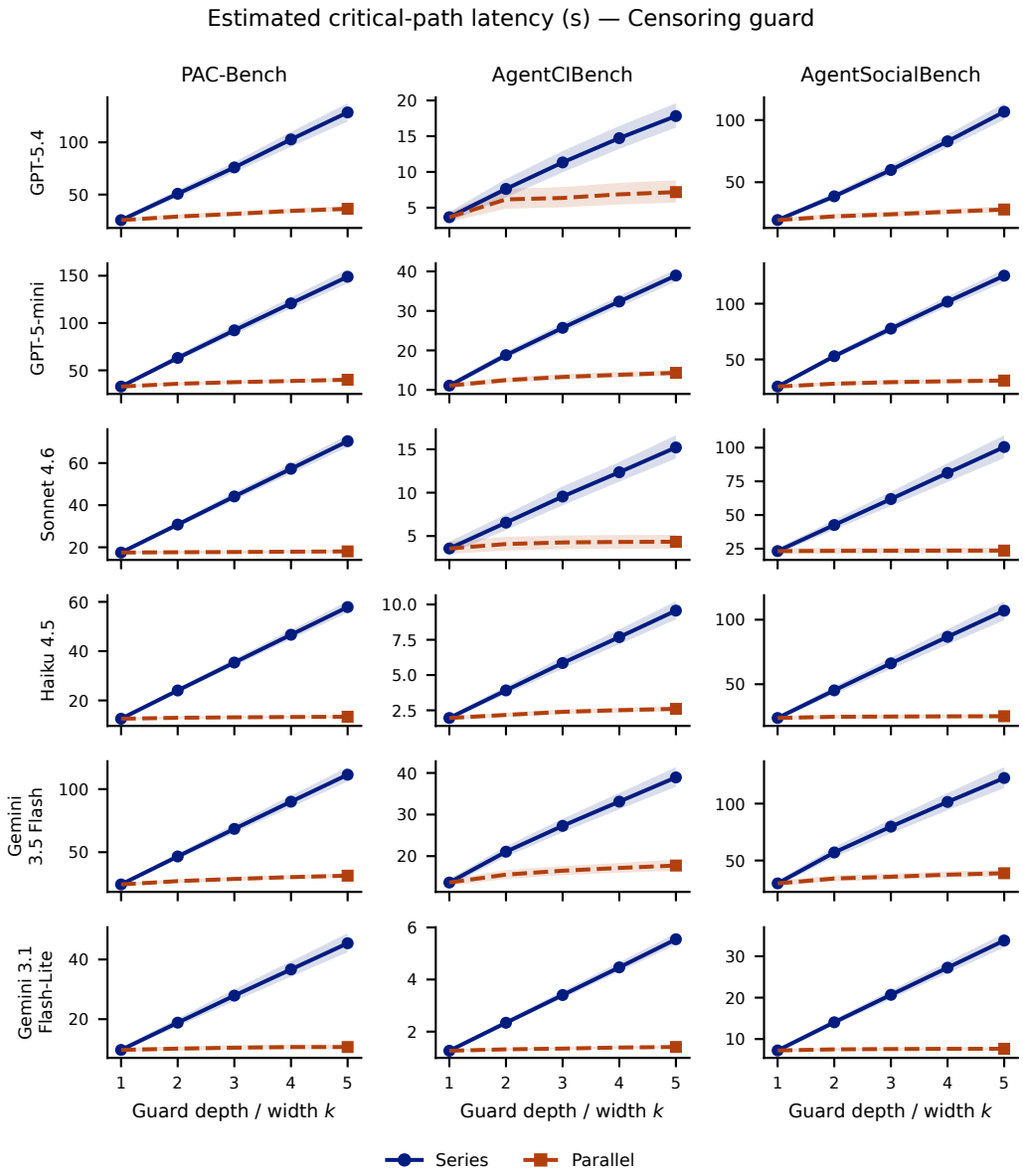

Figure 13: Estimated critical-path latency against the number of guards k for the censoring guard, with one row per guard model and one column per dataset. Bands are 95% bootstrap intervals clustered by scenario.

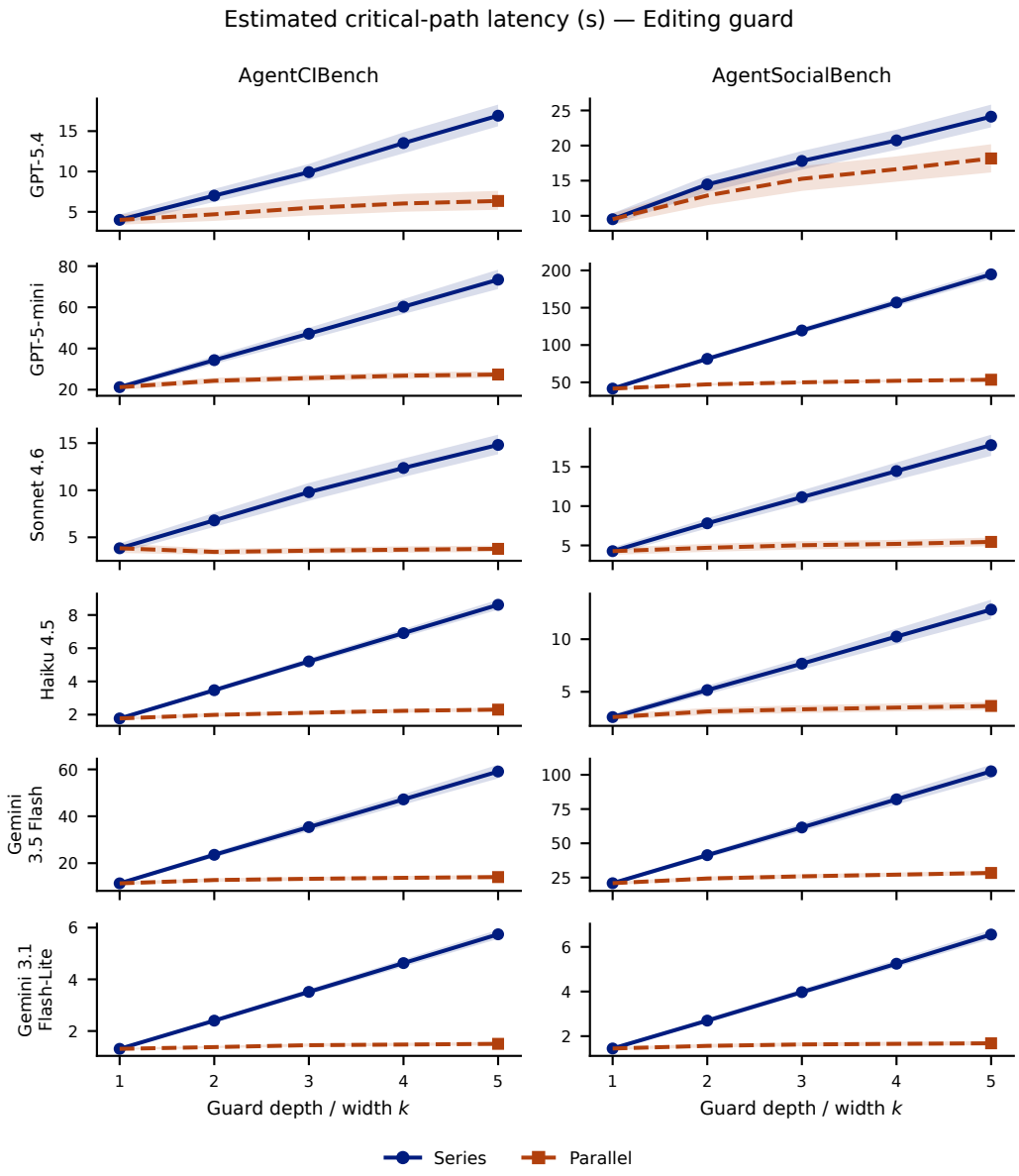

Figure 14: Estimated critical-path latency against the number of guards k for the editing guard. Same layout and intervals as Fig. 13.

F LIMITATIONS AND SCOPE

Context dependence. A primitive signature is valid only for its declared operating regime and input conditioning. An empirical average is not a uniform conditional bound, and reusing the same model does not ensure transfer after an upstream replacement.

Dependence. Nominal fork–join formulas can fail when branch independence or the calibrated selector-response tables drift under shared model weights, prompts, providers, or retrieved context. Marginal-only bounds are always available but may be loose.

Semantic measurement. The tests Rec_q and Use_d are application-specific. The theory is only as meaningful as the evaluators and threat model used to define them.

Selector-response abstraction. The selector receives full messages; we use ψ and ϕ as probability-valued response tables, not runtime inputs. Exact scalar reuse requires summary compatibility and stability under the proposed upstream change. A deterministic content-aware selector need not factor through the Boolean branch labels, and separate privacy/utility tables need not specify a jointly safe-and-useful output. We must distinguish selection, redaction, abstention, and synthesis in calibration; the rare-error theorem additionally excludes a fixed all-useful task-failure floor.

Guard-only scalar abstraction. Reasoning agents and tools can create new sensitive inferences. Modeling them compositionally requires richer boundary-transfer laws than the scalar survival signature used here.

General topologies. The exact local calculus applies to typed series-parallel circuits. A cross-branch bridge can change global reachability without changing the local component signatures, so general dense graphs require richer boundary summaries, bounds, simulation, or a separate optimizer. Exact conditioning on a small number of designated bridge edges remains possible but is not part of the local reduction calculus.

Structural channel completeness. We must represent logs, memories, tool outputs, exception traces, and side channels whenever they can transmit protected information.