



Agent Foundry: A Boundary-Preserving Architecture for Governed Agentic Platforms

Arghya Mukherjee

Independent Researcher.

To Cite this Article: Arghya Mukherjee, “Agent Foundry: A Boundary-Preserving Architecture for Governed Agentic Platforms” Indian Journal of Computer Science and Technology, Volume 05, Issue 03 (September-December 2026), PP: 224-242.



Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the [Creative Commons Attribution License](#); Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: Agent frameworks expose tools, delegation, memory and human approval, yet the controls attached to a request (identity, tenant, scopes, budgets, deadlines, rate limits, fail behavior) are routinely lost at execution boundaries: delegation, threads, streaming, resume, caching, retrieval and adapter exports. We present Agent Foundry, an open-source Python platform whose architecture is organized around one principle: every control is a property of the request’s execution scope, not an argument that each layer must remember to pass. We formalize this as the Governed Execution Contract (GEC), a meet-semilattice of control scopes with seven invariants, and prove that delegation can only attenuate authority, that deny rules can only tighten resolution, that per-principal rate limits are exact under arbitrary thread interleavings, that every call yields exactly one audited decision even when the policy source is unreachable, and that admission partitions by principal without cross-shard coordination. On AgentGovBench, a public 48-scenario gov-ernance benchmark, Agent Foundry passes 48/48 in 10 of 10 repetitions, against 46/48 for the best published governed configuration and 13/48 for each of 7 pub-lished framework-native runners; run through the same harness, Microsoft’s Agent Governance Toolkit scores at most 46/48 and Bounded Agents 36/48, and on a 24-scenario suite authored blind to our code Agent Foundry scores 24/24 against 17 and 8. A set of 13 single-control ablations attributes 35 of the 48 scenarios to specific mechanisms and exposes a measurement problem: a runner that denies every call scores 31/48, above the audit-only reference, and two controls can be re-moved without losing an official point. Eight supplemental discriminating scenar-ios and three anti-vacuity checks close that gap. A 360-degree evaluation covers the remaining layers: multi-agent coordination overhead is $5.6\text{--}154\times$ lower than LangGraph’s on identical topologies; the control plane adds $14\ \mu\text{s}$ per call over the tool gateway and scales to 25,172 governed calls/s across 8 principal-affine shards with exact limits; budgeted context assembly retains a planted fact in 100% of cases up to 1,426,941 history tokens where a recency window retains 20%; a trained input gate raises held-out prompt-injection recall from 0.0% to 80.0%. On WorkBench, matched Foundry runtimes are statistically indistinguishable from the benchmark’s reference loop with one model; we do not claim task-level state of the art. All results regenerate from committed raw evidence.

I. INTRODUCTION

An agentic platform turns one user request into many executions. An orchestrator delegates to specialists; specialists fan out to workers on threads; a run pauses for approval and resumes in another process; a cache answers a repeated prompt; a retriever injects documents into context; a tool is exported to a different framework. The request, however, arrived with controls attached: who the caller is, which tenant owns the data, which scopes were granted, how much may be spent, how many calls per minute are allowed, and what to do if the policy service is down. Each boundary above is a place where those controls can silently disappear.

This is not hypothetical. In a source review of Agent Foundry before this work, request deadlines, cancellation and deny-all tool policies were ignored at the public API in both of its runtimes; a viewer’s request routed through a native supervisor executed an admin-only tool while the Lang-Graph runtime correctly denied it; a human approval overrode an external policy engine that was never consulted; and a prompt cache ignored tool schemas (§6.4). Public evidence shows the same failure class across the ecosystem: on AgentGovBench [1], each of seven widely used frameworks and coding agents scores 13/48 when run without an external governance layer: as run, none propa-gates the end user’s identity into an audit record, narrows delegated scope, or aggregates rate limits across subagents.

We argue that these are not independent bugs but consequences of an architectural choice. Many frameworks treat controls as *arguments*: values that each layer must receive and forward. Every new boundary is then a new opportunity to forget one. Agent Foundry instead treats controls as a property of the *execution scope* of a request. Scopes compose by a meet operation, so nesting can only narrow them; live handles (budgets, cancellation tokens) are kept out of persisted state; and a single deny-first admission point decides and audits every tool call. We call the resulting specification the *Governed Execution Contract* (GEC).

Research questions. **RQ1** (foundations): can the control semantics an enterprise platform needs be stated as a small algebra whose laws are provable and machine-checkable? **RQ2** (governance): does a boundary-preserving architecture meet a public governance benchmark, and which mecha-nisms are responsible for which outcomes? **RQ3** (measurement): do existing governance bench-marks distinguish correct enforcement from trivial behavior? **RQ4** (cost and breadth): what do the controls cost, and how does the platform perform on the other layers—orchestration, context, security filters, evaluation, task execution and scale?

Contributions.

1. A formal contract. The GEC (§3) models control scopes as a meet-semilattice and states seven invariants. We prove

attenuation along arbitrary delegation chains, deny-monotonicity of policy resolution, exactness of per-principal rate limits under arbitrary interleavings, decision totality under policy-source failure (including an impossibility argument for reading the fail mode from the failed source), and coordination-free sharding by principal. Property-based tests check the algebraic laws on randomly generated policies and chains.

2. **A platform architecture.** Agent Foundry (§4) realizes the contract across seven layers: declarative entry points, a think–act–critique orchestration loop, a runtime seam with a native engine and a LangGraph engine, a governed tool gateway, a multi-tenant control plane, an LLM gateway, and a budgeted context layer, with evaluation and observability as cross-cutting rails. We describe the design decisions that differ from common practice and why.
3. **State-of-the-art governance results with mechanism attribution.** Agent Foundry passes 48/48 AgentGovBench scenarios in 10/10 repetitions, above every published configuration (best 46/48) and above the two closest open-source systems run under our harness (Agent Governance Toolkit at most 46, Bounded Agents 36); on 24 scenarios written blind to our code it scores 24/24 against 17 and 8. Single-control ablations (13) attribute 35 scenarios to specific mechanisms (§6.1).
4. **A measurement finding and reedy.** Extending trivial-agent sanity checks [77] to an en-forcement benchmark: Constant policies pass most individual scenarios (44/48 are passed by at least one trivial runner); a deny-everything runner scores 31/48; and 2 controls (delegation attenuation, deny-overrides resolution) can be removed without losing a point. Published runs corroborate this: 6 of 8 governed runs pass the task-narrowing scenario while declaring that they do not implement task narrowing. We contribute eight discriminating scenarios and three anti-vacuity checks (§6.2).
5. **A 360-degree evaluation** across orchestration overhead, governance overhead, thread and process scaling, long-context assembly, public injection and hallucination datasets, and matched WorkBench task execution (§6.5), with negative results kept: the default marker-based injection filter detects 0.0% of held-out attacks (we add a trained gate that detects 80.0%), keyword retrieval fails paraphrased queries, and task accuracy does not improve significantly over the benchmark’s reference loop.

Scope of claims. We claim state of the art only where we measure it against published results under the same scorer: governance on AgentGovBench, and coordination overhead against LangGraph on identical topologies. We do not claim task-level superiority on WorkBench, where the best published model run reaches 674/690 with a stronger model and a different harness, nor multi-agent *quality* advantages, which require model-driven benchmarks we did not run. The control plane was developed during this study with knowledge of AgentGovBench’s categories; we treat this as a threat to validity and address it with ablations, self-authored discriminating scenarios, and anti-vacuity checks rather than by assertion (§7.1).

Artifact. Code, evidence and this manuscript are at <https://github.com/arghya05/agent-foundry>. Every number in this paper is generated by `research-paper/build_evidence.py` from raw files under `review/evidence/`; the main campaign was run from commit 730be18 and produced 158 files with SHA-256 digests in its manifest.

II. RELATED WORK AND RESEARCH POSITION

We compared Agent Foundry with about forty systems and papers; Table 1 summarizes the closest, and Appendix F gives the per-invariant matrix with source or section pointers. Framework code was inspected at pinned revisions.

Agent architectures and frameworks. ReAct [73], Reflexion [57] and Self-Refine [40] established the reason–act and self-critique loops that our think–act–critique cycle follows; CoALA [61] supplies the memory/action/decision vocabulary our layers map onto. AutoGen [68], MetaGPT [24], CAMEL [34] and multi-agent debate [20] motivate our six topologies, and MAST [7] catalogues their failures. None of these is our contribution. Production frameworks provide the *mechanisms* of pause, resume and approval: LangGraph [33] persists interrupts and implements cooperative timeouts; Pydantic AI [53] wraps approval-required toolsets and declares per-engine durability; CrewAI [12] persists flows but drops callable guardrails on serialization; Microsoft Agent Framework [46] matches approval rules to tools and arguments; Deep Agents [32], the OpenAI Agents SDK [51] and Google ADK [21] offer subagents, guardrails and plugins. Their own documentation or code shows the boundary problem this paper addresses: subagent budgets and rate buckets are per agent rather than per principal (ADK’s agent tool, Microsoft Agent Framework’s budget keys); Deep Agents’ subagent permissions replace the parent’s and default to allow; and approval takes precedence over conflicting sticky decisions in the OpenAI SDK. Their AgentGovBench native rows are 13/48.

Delegation and authorization for agents. Multi-hop attenuation by intersection is established. Bounded Agents [49] attenuates scope along a principal chain and adds composition closure, delegation budgets, single-use hash-bound approvals and intent binding, reducing AgentDojo exfiltration to zero at a utility cost of 8.6–13.9 points. Bounded Agents also states cumulative consumption as an invariant—each child inherits its parent’s consumed budget as a floor [49]—and CapLease charges a delegated child against its parent’s authorization budget [70]; these are our I4 for a single chain or a single authorization, and our addition is keying shared buckets by principal and tenant across sibling threads with exact windows under concurrency (Theorem 3). Further, CAPMAS [64] and Overlaying Governance [25] formalize capability and compositional delegation; Delegation Without Trust [14] finds LangGraph, CrewAI and AutoGen exposed to confused-deputy and escalation threats and proposes a token broker; MiniScope [76], SEAgent [28], authenticated delegation [59] and ScopeGate [42] treat least privilege, mandatory access control, OAuth-style delegation and the gap between exposing a tool and authorizing it. Forbid-first combination is also established: Progent [56] evaluates forbid rules before allow rules with a default block, and ActGov [75] and the Agent Governance Toolkit [45] use deny-overrides. Per-principal budgets across agents exist for risk (Irreversibility Budget [48]), and MasuGate [52] makes budget checks safe under concurrent effects via policy-state serializability. Liu [38] names eight security invariants for MCP-style runtimes (principal binding, scoped invocation, data-flow authorization, deny-path audit, among others) and Tallam [62] four for a governance reference architecture. We therefore claim neither attenuation, deny-overrides, per-principal budgeting nor a named invariant set as new. What we did not find in prior work is the *combination* realized as one execution-scope object in a framework runtime; complete cache identity (I5) as a governance invariant; the non-serialization of live budget and cancellation handles as

the persisted half of I3; measurement honesty (I7) stated as an obligation of the runtime’s own evaluation layer rather than only of benchmark reporting [58, 77]; a fail mode that a tenant can set to open and that is known during a policy-source outage, with Proposition 2—existing systems either fail closed unconditionally [45, 49] or let individual advisory components declare a local error behaviour [45]; and the benchmark results below. Several designs take the opposite position to I3. CapLease [70] shows that single-use identifiers cannot bound consumption across retries and crash recovery without durable token-independent state; ResidualAuth [10] proves that current reachability can collapse exponentially many future-distinct authorization states under revocation; the Agent Governance Toolkit persists pending approvals across restarts and revalidates the action digest and policy version immediately before execution [45]; and MasuGate shows that revalidation at resolution is correct but can invalidate a delayed human approval, which durable reservations avoid [52]. In ResidualAuth’s terms our re-derivation on resume *externalizes* the query to the live policy source rather than preserving authorization state; it therefore cannot express a spent one-shot approval, and it is sound only while that source is available—the case Proposition 2 addresses. We discuss this trade-off in §4.2. MasDrift [71] measures up to 36.3 points of task-completion loss from strict attenuation across multi-agent architectures; we do not measure this utility cost.

Content and information-flow security. CaMeL [15] and FIDES [11] track information flow and attach capabilities to data, defending against prompt injection by construction; the design-patterns paper [6] argues for such architectural defenses. These are strictly stronger than our filter-based I6. Classifier and rail systems—NeMo Guardrails [54], Guardrails AI [22], Llama Guard [26], Prompt Guard [43], Invariant [27]—and policy or verifier approaches—AgentSpec [65], AgenTRIM [5], SABER [13], ProbGuard [66], Consecra [63], ShieldAgent [9], GuardAgent [69], DRIFT [35]—decide whether content or an action is acceptable. The GEC is orthogonal to all of them: it guaran-tees that whichever policy or detector is used is evaluated with the right principal, tenant, attenuated scope, shared budget and fail behavior at every boundary.

Governance products and audit. The Agentic Control Plane gateway [2] is the closest compara-tor for our control plane and supplies the strongest published AgentGovBench runs. Microsoft’s Agent Governance Toolkit [45] binds approvals to action digests, policy versions and expiry, con-sumes them once, and hash-chains its audit log; Agent Foundry has neither action-bound approvals nor tamper-evident audit, and we exclude both from our claims. Its policy engine fails closed on every evaluation error with no fail-open option, and its budgets and rate buckets are keyed per agent or globally rather than per principal; MasuGate’s procurement benchmark finds the toolkit’s per-agent cost guard exact but its team-budget and inventory decisions stale under concurrency [52]. AgentBound [31], Auditable Agents [50] and delegation-ancestry attestation [37] strengthen audit cryptographically.

Transactions and recovery. Mnemosyne [8] admits agent actions as transactions against declared constraints with evidence-preserving repair, and CFRC [18] preserves obligations across stateful handoffs. Both address what happens after admission, which the GEC does not.

Evaluation. Task benchmarks—WorkBench [60], τ -bench and τ^2 -bench [72, 3], Agent-Bench [39], GAIA [44]—measure capability; AgentDojo [16] and InjecAgent [74] measure in-jection robustness with a model in the loop; AgentGovBench [1] measures the governance layer without one. Kapoor et al. [30] argue for baseline-controlled, cost-aware agent evaluation. Trivial baselines are an established sanity check. The Agentic Benchmark Checklist asks authors to report trivial agents and finds that a do-nothing agent passes 38% of τ -bench airline tasks [77]; a validity audit of agent-safety benchmarks shows that an always-positive policy outranks five of 21 models on R-Judge [67]; DEMM-Bench reports always-sufficient and always-insufficient anchors for governance-evidence scoring [58]; ResidualAuth scores counterfactual pairs so that a constant al-low or deny policy earns no credit [10]; and CapLease checks that legitimate budgets are consumed, ruling out safety by blanket rejection [70]. Single-component ablation of an enforcement runtime is also established: Liu disables one component at a time on a ten-case benchmark and reports the cases each removal exposes [38]. What we add is applying both to a third-party enforcement benchmark and its published leaderboard at scenario granularity—which scenarios a constant runner passes, which controls no scenario observes, and published runs that pass a scenario for a control they declare they do not implement. Per-control ablation is the benchmark analogue of mutation analysis [29], and a scenario passed by a constant runner is the analogue of a vacuously satisfied property in model checking [4].

Work	Overlap with Agent Foundry	Ahead of Agent Foundry
Bounded Agents, CAPMAS, Overlaying Gov, Progent, ActGov, AGT	Multi-hop attenuation (I1), cumulative budget floor (I4), fail-closed, audit Forbid-first / deny-overrides (I2)	Composition closure, bound approvals, intent binding
Irreversibility Budget, MasuGate, CaMeL, FIDES, Tallam, CapLease, ResidualAuth	Per-principal budgets (I4) Content boundary (I6) Named invariant set (opposite of I3)	Verified synthesis (ActGov); bound approvals, hash-chained audit (AGT) Concurrency-safe budgets across effects Information-flow control Reference architecture Durable authorization state across resume
ACP gateway	Hosted admission, audit	Managed service; 37–46/48 published Pair-level design (ResidualAuth)
DEMM-Bench, ABC, validity audit, Liu MCP invariants	Trivial anchors / trivial agents in benchmarks Per-control ablation (own benchmark)	Source-and-target data-pipe checks
Frameworks (LangGraph, CrewAI, SDKs)	Engines, interrupts, approvals	Ecosystem breadth; 13/48 native

Table 1: Research position. “Ahead” lists capabilities in which the cited work exceeds Agent Foundry.

III. THE GOVERNED EXECUTION CONTRACT

This section states the contract that the architecture implements. The definitions follow the code closely (module and symbol names are given), so that each theorem is a statement about the shipped system rather than an idealization; assumptions that the code does not discharge are stated explicitly.

3.1 Control scopes and attenuation

Let T be a finite set of tool names, S a finite set of scope names, and $A = \{L_0 < \dots < L_5\}$ the autonomy levels. A tool policy is a tuple

$$\pi = (T, R, c, k, a) \in 2^T \times 2^T \times R_{\geq 0} \times N \times A, \quad (1)$$

of allowed tools T , approval-required tools R , a cost ceiling c , a step ceiling k and an autonomy level a (contracts.Policy). Order policies by permissiveness:

$$\pi \sqsubseteq \pi' \iff T \subseteq T' \wedge R \supseteq R' \wedge c \leq c' \wedge k \leq k' \wedge a \leq a'. \quad (2)$$

Definition 1 (Attenuation). The meet of two policies, implemented by `execution_scope.intersect_policies`, is

$$\pi \sqcap \pi' = (T \cap T', R \cup R', \min(c, c'), \min(k, k'), \min(a, a')).$$

Lemma 1. $(\Pi, \sqsubseteq)$ is a meet-semilattice with meet $\sqcap$: the operation is commutative, associative and idempotent, and $\pi \sqcap \pi'$ is the greatest lower bound of π and π' .

Proof. Π is a product of five lattices: $(2^T, \subseteq)$ with meet $\cap$; $(2^T, \supseteq)$ with meet $\cup$; two chains $(R_{\geq 0}, \leq)$ and $(N, \leq)$ with meet $\min$; and the chain A with meet $\min$. A product of meet-semilattices ordered componentwise is a meet-semilattice whose meet is the componentwise meet, and each componentwise meet is commutative, associative and idempotent.

A request scope C additionally carries a principal (u, τ) (user and tenant), a role set $\rho \subseteq S$, a deadline d , a cancellation predicate κ , a model allowlist M , a budget bucket list B , a delegation chain $\chi \in G^*$ over agent names G , and optionally a delegated scope set $\delta \subseteq S$. Nesting (`execution_scope.request_scope` and `control_plane.delegation`) combines a parent C_p with a requested child C_r as

$$C_p \oplus C_r = (\pi_p \sqcap \pi_r, \rho_p \cap \rho_r, \min(d_p, d_r), \kappa_p \vee \kappa_r, M_p \cap M_r, B_p ++ B_r, \chi_p g, \delta_p \cap \delta_r), \quad (3)$$

where components absent from C_r are inherited from C_p . `request_scope.rejects` the nesting if

$(u, \tau)_p \not\models (u, \tau)_r$: a nested execution may not replace its calling principal. delegation changes

only χ and δ , and the control plane receives (u, τ) explicitly on every call, so a delegated hop cannot name a different principal either. We write $\delta = T$ for “no delegated restriction” with $T \cap X = X$.

Theorem 1 (Attenuation along delegation chains). For any chain $C_0, C_1, \dots, C_n$ with $C_{i+1} = C_i \oplus R_i$ and any $0 \leq i \leq j \leq n$: $\pi_j \sqsubseteq \pi_i, \rho_j \subseteq \rho_i, \delta_j \subseteq \delta_i, M_j \subseteq M_i, d_j \leq d_i, \kappa_i \Rightarrow \kappa_j$, and B_i is a prefix of B_j . Consequently the effective scope of a tool call made at depth j , $\text{eff}_j = \sigma(u, \tau) \cap \delta_j$ where σ is the principal’s granted scope set, satisfies $\text{eff}_j \subseteq \sigma(u, \tau) \cap \delta_i$ for the declared sets δ_i , so no descendant can regain a scope dropped by any ancestor.

Proof. By induction on $j - i$. The base case is reflexivity. For the step, each component of Eq. (3) is a meet (or disjunction for κ , append for B) of the parent component with the request component, and a meet is below each argument. Transitivity of $\sqsubseteq$ and $\subseteq$ closes the induction. The bound on eff_j follows by unfolding $\delta_j = \delta' \cap \dots \cap \delta_i$.

The budget component is the one place where nesting does not take a meet: a child with a new thread identifier appends its bucket, and every charge is applied to *all* buckets in B (`_ScopedBudget`). Hence no child can escape the root’s ceiling by renaming its thread (invariant I4).

3.2 Policy resolution and deny precedence

A workspace policy (`control_plane.WorkspacePolicy`) maps a tier $t \in \{\text{interactive}, \text{subagent}, \text{api}, \text{background}\}$ to rules at four specificities: tenant default $D(t)$, user-wide $U_u(t)$, tool-wide $W_x(t)$, and user-and-tool $X_{u,x}(t)$. A rule is a pair (p, ℓ) of a permission $p \in \{\text{allow} < \text{flag} < \text{deny}\}$ and an optional rate limit $\ell \in N \cup \{\infty\}$. Specificity is a partial order: X is above both U and W , which are incomparable, and both are above D .

Definition 2 (Resolution).

$$\text{res}(u, x, t) = \begin{cases} X_{u,x}(t) & \text{if defined,} \\ \max_{<} \{U_u(t), W_x(t)\} & \text{if either is defined (deny-overrides),} \\ D(t) & \text{if defined,} \\ \perp & \text{otherwise (no rule: deny),} \end{cases}$$

with the resolved rate limit $\ell^* = \min$ of the limits of all defined rules among X, U, W, D .

Theorem 2 (Deny monotonicity). Let P' be obtained from P by setting any rule at the U, W or D level to deny, or by adding a rule. If no X rule applies, then $\text{res}_{P'}(u, x, t) \geq \text{res}_P(u, x, t)$ whenever the change is a deny, and the resolved rate limit never increases when a rule with a limit is added.

Proof. If the modified level is U or W , the resolved permission is a maximum over a set that now contains deny, which is the top of the chain. If the modified level is D and some U or W rule is defined, the resolution is unchanged; otherwise it becomes deny. Adding a limited rule adds an element to the set over which ℓ^* is a minimum.

The only way to loosen a resolution is an explicit $X_{u,x}$ rule, which names both the principal and the tool; this is the “most specific wins” exception that AgentGovBench’s scenario `per_user_policy_enforcement.03` requires. A user-wide allow therefore cannot override a tool-wide deny (our supplemental scenario 03 tests exactly this). Among the four resolution branches, the absence of any rule denies, so an unconfigured tier fails closed.

3.3 Deny-first admission

Admission (`ControlPlane.invoke`) evaluates, in order: (1) authentication ($u \neq \varepsilon$); (2) policy-source availability; (3) tenant existence and membership $u \in \text{members}(\tau)$; (4) tool registration; (5) resolution $r = \text{res}(u, x, t) \neq \perp$; (6) required scopes $\text{req}(x) \subseteq \text{eff}$; (7) $r.p \neq \text{deny}$; (8) the rate limit; then execution through the governed gateway, which re-checks allowlist, scopes, budget and approval (`GovernedToolGateway`). The first failing check determines the reason.

Proposition 1 (Totality and one decision per call). *Every invocation terminates with exactly one decision in {allow, flag, deny} and appends exactly one governance_decision audit record, in-cluding when the policy source raises, when the gateway denies, and when the tool fails.*

Proof. Each branch of `invoke` and `_execute` returns the value of `_record`, which appends one record and constructs the decision. `PolicySourceUnavailable` is caught at step (2); in `_execute`, `PermissionDenied` and every other exception raised by the gateway (budget exhaustion, cancellation, deadline, unexpected errors) are converted into a recorded denial; a tool that fails after admission yields an allowed decision with a failed result. No other path returns. The argument assumes that the audit sink itself does not raise.

Proposition 2 (The fail mode must be local). *Let the decision under an unavailable policy source depend on a tenant-configured fail mode $f_\tau \in \{\text{open}, \text{closed}\}$. If f_τ is stored only in the policy source, then for a tenant whose first request arrives during an outage, no admission procedure can honor $f_\tau = \text{open}$.*

Proof. During the outage the procedure’s inputs are the request and local state. If f_τ is stored only remotely and has never been read, local state is identical for $f_\tau = \text{open}$ and $f_\tau = \text{closed}$, so the procedure returns the same decision in both worlds; it must deny for closed, hence cannot allow for open.

The first version of our control plane learned f_τ from successful reads and failed AgentGovBench scenario `fail_mode_discipline.02` for exactly this reason; the shipped version takes f_τ as deployment configuration and refreshes it on reads (ablation “locally declared fail mode” in Table 5).

3.4 Rate limits under concurrency

Let calls for key $\kappa = (\tau, u, t)$ be admitted by `SlidingWindowLimiter.try_acquire`, which under a mutex drops timestamps older than W and admits iff fewer than ℓ remain.

Theorem 3 (Exactness under arbitrary interleavings). *Assume a monotone clock and that each call’s check-and-append executes under the mutex. For any finite set of concurrent calls on key κ and any interleaving: (safety) every interval of length W contains at most ℓ admissions; (liveness) if n calls reach the limiter within one window and no earlier admission for κ lies in that window, exactly $\min(n, \ell)$ are admitted. Calls denied at steps (1)–(7) do not consume capacity.*

Because each check-and-append is atomic, the execution is equivalent to a sequential history ordered by lock acquisition (linearizability of a critical section). In that history, the deque for κ contains exactly the admissions within the trailing window at each step, and an admission occurs iff its size is below ℓ ; so the count in any window never exceeds ℓ (safety), and the first $\min(n, \ell)$ calls in lock order are admitted while the rest see a full deque (liveness). Steps (1)–(7) precede the limiter and return before `try_acquire`.

One consequence is worth stating: a call that passes the limiter but is then denied by the gateway (for example by a second-line scope check or an exhausted step budget) has consumed capacity. Checks that can deny should therefore precede the limiter, which is why the plane re-implements the scope check the gateway also performs. The assumption of a monotone clock matters. The first campaign used wall-clock time for windows and recorded one repetition, out of 6×3 , in which a principal was not admitted exactly ℓ times; the anomaly did not reproduce in 30 reruns. A wall-clock adjustment can shift a window, so the shipped limiter uses `time.monotonic`; all 10 repetitions at every thread count in the later campaigns are exact (§6.5).

3.5 Partitioned admission

Proposition 3 (Coordination-free sharding). *All mutable admission state (limiter deques, learned fail modes) is keyed by (τ, u, t) or τ , and policy documents are read-only per call. If requests are routed by a function of (τ, u) , then K independent control planes, each holding the policy of its principals, produce the same decisions as one control plane, with no messages between shards; aggregate throughput is bounded above by K times single-shard throughput.*

Proof. A decision for (τ, u, t, x) reads the policy of τ , the membership and scopes of u , and the deque for (τ, u, t) , and writes only that deque and one audit record. Routing by (τ, u) places all reads and writes for a key on one shard, so each shard’s execution is a restriction of the single-plane execution to its keys, and the union of the restrictions equals the whole.

Principal-affine routing is a deployment pattern, not a library component: `benchmarks/sharded_scaling.py` implements the router used in §6.5. Budget buckets shared across principals (for example a tenant-wide cost ceiling) are the exception to the proposition: they need a shared counter, which the `RunBudgetLike` protocol permits but which we do not evaluate. Each bucket’s charge is atomic under its own lock, but a charge applied to several ancestor buckets is not one transaction; `MasuGate` [52] addresses budget consistency under concurrent effects.

3.6 The seven invariants

Collecting the above, the GEC requires: **I1** attenuation (Thm. 1); **I2** deny precedence—hard denials precede approval

requests in the PDP, and resolution is deny-monotone (Thm. 2); **I3** live/persisted separation—budget and cancellation handles are never serialized, and resume re-derives controls from the current request; **I4** shared accounting across children (Thm. 1, budget clause; Thm. 3); **I5** complete cache identity—a prompt-cache key digests the full effective request, including tools, call identifiers, sampling options and response format, and the calling tenant and user from the scope; **I6** a uniform content boundary—user input, retrieved passages and tool results pass the same injection detectors (retrieved passages flagged by them are dropped and PII-redacted; flagged tool results are labelled untrusted rather than dropped, because they are often the data the caller asked for); and **I7** measurement honesty—a missing oracle is unmeasured rather than passed, an empty gate fails, and failed or interrupted attempts remain in denominators with their costs. I3, I5, I6 and I7 are implementation obligations checked by regression suites (§6.4) rather than theorems.

3.7 Measuring whether a benchmark measures

Let B be a benchmark of scenarios s with pass predicate $\text{pass}(s, o)$ over observed outcomes o . Let R_0 be a set of *trivial runners*: constant or enforcement-free policies (no governance, audit-only, deny-everything, allow-and-audit).

Definition 3. Scenario s is *vacuous* if $\exists r \in R_0$ with $\text{pass}(s, o_r)$. Given a system F and single-control ablations F_{-c} , s is attributed to c if $\text{pass}(s, o_F) \wedge \neg \text{pass}(s, o_{F-c})$. A control c is unobservable in B if no scenario is attributed to it.

A benchmark score is informative about a mechanism only through scenarios attributed to it; an unobservable control can be deleted without changing the score. We complement pass predicates with three *anti-vacuity* checks computed on outcomes: *decision coverage* (one audited decision per attempted call), *effect fidelity* (executed effects equal allowed outcomes) and *liveness* (a scenario that expects any allowed or rate-limited call admits at least one call).

IV. ARCHITECTURE

Agent Foundry is a Python package (`agent_foundry`, 61 modules) whose base install depends only on cryptography; every integration (LangGraph, LangChain, model providers, MCP, Redis, Postgres, Chroma, OPA/Cedar, OpenTelemetry, LiveKit) is an optional extra. Figure 1 shows its layers. We describe each by its responsibility, its interface, and the GEC obligation it discharges, then explain the design decisions that differ from common practice.

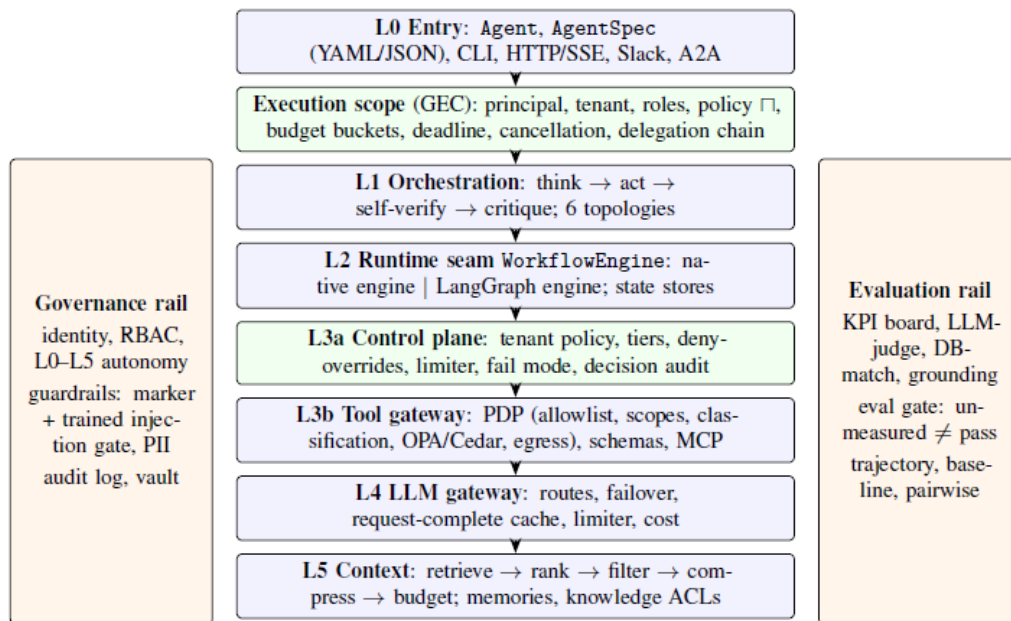


Figure 1: Agent Foundry layers. Green boxes carry the GEC: the execution scope is established once at the entry point and consumed by every lower layer; the control plane is the single admission point for tool effects. Rails are cross-cutting services invoked from several layers.

4.1 Layers

L0 Entry and configuration. An agent is defined once, imperatively (`Agent(...)`) or declaratively (`AgentSpec` from YAML/JSON with structured tool metadata and named critique evaluators), and exposed through `run/arun/stream/resume/batch`, a CLI, an HTTP/SSE server with a browser console, a Slack channel and an A2A bridge. Every public entry point is wrapped by `scoped_execution`, which converts an `ExecutionContext` into the request scope of §3. This is where controls enter; lower layers read them from the scope rather than from parameters (direct graph-state callers retain an older state-key interface, which the resolvers also honor).

L1 Orchestration. A turn runs *think* (prompt assembly and a model call), *act* (tool calls, dispatched concurrently where independent), an optional *self-verify* revision, and *critique*, a KPI-scored gate that retries with more evidence, asks a clarifying question, or escalates to a human. Six multi-agent topologies—supervisor, swarm (handoffs), blackboard, debate, fan-out and

DAG—reuse the same per-agent configuration, so governance, budgets and critique apply to every specialist.

L2 Runtime seam. Execution is a WorkflowEngine protocol (build, run, stream, resume) selected from a registry. Two engines implement it: a from-scratch native loop, and a LangGraph StateGraph engine for LangGraph’s checkpointing and interrupts. The same configuration runs on either, which enables the differential testing of §6.4. Native single-agent state persists through versioned StateStores (in-memory, SQLite, Redis, Postgres) with optimistic version checks.

L3 Tool effects. The *control plane* (control_plane.ControlPlane) is the multi-tenant ad-mission point of §3: tenant policy documents re-read per call, tier-aware rules with deny-overrides, per-principal sliding-window limits, attenuating delegation, a locally declared fail mode, and one audit record per decision. Admitted calls execute through the *governed tool gateway* (GovernedToolGateway), which requires a trusted user and tenant on every call, re-applies the policy decision point (allowlist, declared scopes, data classification, optional OPA or Cedar engine, egress allowlist, approval), enforces budgets and JSON-Schema validation of inputs and outputs, binds idempotency keys to principal and payload, and audits admission and results. MCP servers’ tools enter as ordinary tool specifications and are governed identically.

L4 LLM gateway. Tasks route to ranked model lists with failover, a prompt cache whose key digests the complete request and the calling principal (I5), a rate limiter and per-call cost accounting. Model allowlists in the request scope narrow the candidate list; an empty intersection is an error, not a fallback to any model.

L5 Context. Working, episodic, semantic, procedural, knowledge-graph and profile memories sit behind small protocols. A ContextEngine assembles prompt context by retrieval, ranking, filtering, compression and token budgeting. Retrieved passages and tool results are screened by the same injection detectors as user input; flagged passages are dropped and PII-redacted, flagged tool results are labelled untrusted (I6); knowledge chunks carry permission tags filtered by the caller’s roles, and the tenant is taken from the request.

Rails. The *governance rail* provides identity, role-based tool access, six autonomy levels, in-put/output/action/runtime guardrails—including the trained injection gate of §6.5—and audit sinks. The *evaluation rail* provides a KPI board with LLM-judge, database-match and grounding scorers, and an evaluation gate that scores an agent against a versioned dataset and can fail a release; missing oracles are recorded as unmeasured and an empty gate fails (I7).

4.2 Design decisions and their origin

We state which ideas are borrowed and which choices are, to our knowledge, specific to this design; §2 compares them with the closest systems.

Controls as scope, not arguments. Capability systems [19, 47] and the principle of least privilege [55] are old; the confused-deputy problem [23] is exactly the delegated-identity failure we observed. Our contribution is to apply them *at every agent execution boundary*—sync, async, streaming generators, resume, threads, fan-out, DAG, batch, adapter exports—through one context-variable scope with a meet operation, so that adding a new boundary requires wrapping it, not threading a new argument through each layer. Streaming generators re-enter the scope at every step so that a consumer’s own context is restored between yields.

Live handles are not state. Checkpointing frameworks persist graph state; if a budget object or cancellation token is persisted, resume either loses it or resurrects a stale one. We keep them out of checkpoints and require callers to re-supply controls on resume, so a revoked role takes effect on the resumed run. This trades convenience (controls must be supplied again) for the property that per-sisted state can never widen authority. It also has a cost that CapLease [70] and ResidualAuth [10] make precise: state that must *narrow* authority across resume, such as a consumed single-use approval or a revocation history, cannot be re-derived from the request and needs a durable, monotone store. Agent Foundry does not yet provide one; its approvals are re-checked on resume against current policy but are not single-use.

Deny-first, single admission. A decision pipeline that checks approval before hard policy lets approval override policy; we observed this in the original code. The PDP and the control plane evaluate unconditional denials before approval requests, and rate limiting is evaluated last so that denied calls do not consume capacity (Thm. 3).

Fail mode is deployment configuration. Proposition 2 shows that a fail mode stored only in the policy source cannot be honored for a tenant first seen during an outage. We found this empirically before proving it.

Unmeasured is not passed. Evaluation harnesses often default a missing oracle to success or drop failing cases. Agent Foundry’s gate distinguishes *failed*, *unmeasured* and *passed*, and our benchmark harnesses keep runner exceptions and interrupted attempts in fixed denominators. The same principle motivates the anti-vacuity checks of §3.7. DEMM-Bench applies the same rule to benchmark reporting, recording unexercised axes as not evaluated rather than as passes [58].

Engine neutrality as a test oracle. Supporting two engines is common. We use the second engine as a *differential oracle*: identical scenarios run against both, and any divergence in control outcomes is a defect in one of them, as in differential testing [41]. This found the native delegated-identity bug that single-engine tests had not.

Layer / property	Benchmark	Primary metric	Comparator
Governance (L3)	AgentGovBench v0.2, 48 scenarios, unmodified scorer	scenarios passed; 10 repetitions	15 published runs; AGT and Bounded Agents run by us; trivial runners
Mechanisms	13 single-control ablations	scenarios flipped	full system
Measurement validity	8 supplemental scenarios (ours); 24 blind scenarios; anti-vacuity checks	discrimination, coverage, fidelity, liveness	trivial runners; competitors
Boundary conformance (L0–L3)	regression suites across both engines	failing→passing cases	pre-repair code
Orchestration (L1–L2)	8 topologies, scripted model1, 5 runs	p50/p95 latency, memory	LangGraph engine
Governance cost	per-call microbenchmark, 7 reps × 2,000 calls	μs per call	bare call, registry, gateway
Scalability (L5)	1–32 threads; 1–8 shards	calls/s; p99; exact limits	single thread/shard
Context (L5)	long-history needle assembly (ours)	retention within budget	recency window; full history
Security filter	deepset/prompt-injections; AgentDojo tool outputs (model-free)	held-out recall, precision, FPR	marker filter; constants
Hallucination scorer	HaluEval-QA, 10,000 pairs	AUROC; length-matched AUROC	answer length
Task execution (L0–L5)	WorkBench, matched 3-arm study	official accuracy; McNemar	reference loop; published runs

Table 2: 360-degree evaluation map. “Ours” marks deterministic benchmarks built for this study.
Model calls: only WorkBench uses a model.

V.EVALUATION PATROL

Table 2 maps each layer to the evidence we collect. We prefer public benchmarks with published comparators and unmodified scorers; where none exists we build a deterministic benchmark, state that it is ours, and include a naive baseline. Model-free experiments were run from a clean commit by one script (benchmarks/run_paper_campaign.py), which refuses to reuse an output directory and records SHA-256 digests of every output. Four such campaigns were run and all are retained: the first exposed the clock anomaly of §3; the second followed the monotonic-clock fix; the third followed the principal-partitioned prompt cache (I5) and the conversion of gateway failures into audited denials (Proposition 1); the fourth adds the blind suite, the measured competitors and the AgentDojo study without changing Agent Foundry’s code, and supplies every reported number.

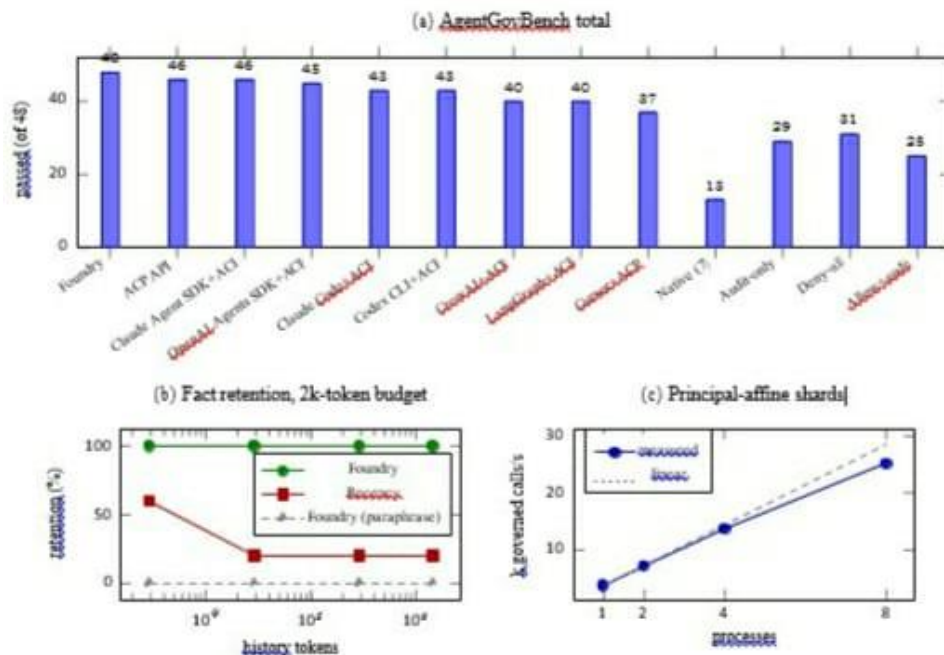


Figure 2: (a) AgentGovBench totals for Agent Foundry, the eight published governed runs, the seven identical framework-native runs, and trivial runners; the deny-everything probe scores above the audit-only reference. (b) Retention of a planted fact by assembled context within a fixed budget as history grows (Table 9). (c) Aggregate admission throughput with exact per-principal limits (Table 8).

AgentGovBench harness. We pin AgentGovBench to commit e0ce93a and import its loader, scorer and aggregation unchanged. Our harness replicates the upstream command-line loop with one stricter rule: a runner exception becomes a failed scenario rather than being skipped, so the denom-inator is always 48. The Foundry runner (benchmarks/agentgovbench/foundry_runner.py) translates each scenario’s tenants, users, tools and policy into Foundry objects and replays its ac-tions through ControlPlane.invoke; it makes no allow/deny decision itself and reads the reported audit log back from Foundry’s AuditLog. Two harness choices are disclosed in its metadata: a vir-tual clock advances past outages of at most 10 s, as the upstream ACP runner does by sleeping, and fan-out workers run on real threads, whereas the upstream reference runners loop sequentially. We reproduce the published vanilla and audit-only baselines through the same harness to check its fidelity.

WorkBench study. The task-level study predates this paper’s control plane and is reported from its archived evidence. WorkBench [60] was pinned to 49c7dfd with the official v2 scorer. Three arms—the benchmark’s reference structured loop, the Foundry native engine and the Foundry Lang-Graph engine—received the same tools, instructions and model (gpt-5.4-mini, low reasoning effort, 2,048 output tokens, at most 20 calls, serial tool dispatch, one repetition). Task sets were sampled per domain with a fixed seed; the first 60-task set was later inspected for failure diagnosis, so only the fresh second set is untouched. Failed and interrupted attempts stay in the denominator; nothing was selectively retried. Costs are upper estimates from a pre-reserving ledger capped at \$25.

Statistics. Paired binary outcomes use the exact two-sided McNemar test; proportions use Wil-son 95% intervals; AUROC intervals use the Hanley–McNeil standard error. Timing results report medians across repetitions. Deterministic benchmarks are repeated to establish that outcomes are identical, not to estimate a variance.

VI.RESULTS

6.1 Governance: AgentGovBench

Table 3 and Figure 2a place Agent Foundry among all published AgentGovBench runs. Agent Foundry passes all 48 scenarios; the result is identical across 10 repetitions, including the concurrent fan-out scenarios. The strongest published configurations reach 46/48; they lose the two cross-tenant scenarios that require a multi-tenant deployment, which their runners declare unsupported. Every framework evaluated without a governance layer scores 13/48. Our harness reproduces the published vanilla (13) and audit-only (29) baselines exactly, which checks that its stricter error handling does not change scores.

Table 3: AgentGovBench v0.2 (48 scenarios, 6 per category, unmodified scorer). Published rows are recounted from the upstream result files at commit e0ce93a; Agent Foundry and the probes are run by our harness. Column abbreviations: Aud audit completeness, Ten cross-tenant isolation, Del delegation provenance, Fail fail-mode discipline, Id identity propagation, Pol per-user policy, Rate rate-limit cascade, Sco scope inheritance. ^aLangGraph, CrewAI, OpenAI Agents SDK, Claude Agent SDK, Claude Code, Codex CLI and Cursor; all seven native runs are identical.

System	Aud	Ten	Del	Fail	Id	Pol	Rate	Sco	Total
Agent Foundry (this work)	6	6	6	6	6	6	6	6	48
ACP gateway (API mode)	6	4	6	6	6	6	6	6	46
Claude Agent SDK + ACP	6	4	6	6	6	6	6	6	46
OpenAI Agents SDK + ACP	6	4	6	6	6	6	5	6	45
Claude Code + ACP	6	4	6	4	6	6	5	6	43
Codex CLI + ACP	6	4	6	4	6	6	5	6	43
CrewAI + ACP	6	4	2	6	6	6	6	4	40
LangGraph + ACP	6	4	2	6	6	6	6	4	40
Cursor + ACP	6	4	4	3	5	5	4	6	37
Seven frameworks, no governance layer ^a	1	4	0	3	0	1	3	1	13
Audit-only reference runner	5	4	5	4	6	1	3	1	29
Deny-everything probe (ours)	6	3	6	3	2	1	5	5	31
Allow-and-audit probe (ours)	5	4	5	4	2	1	3	1	25

Table 4: Head-to-head under one harness and scorer. Coverage: scenarios with exactly one audited decision per attempted call (official suite).

System	Official (48)	Supplemental (8)	Blind (24)	Coverage
Agent Foundry	48	8	24	48/48
MS Agent Governance Toolkit	45	5	16	48/48
AGT, workers share user identity	46	6	17	48/48
Bounded Agents (APC)	36	3	8	41/48
No governance (vanilla)	13	0	0	0/48
Audit-only	29	0	1	47/48

Measured head-to-head. Published rows were produced by their authors. To compare mechanisms under identical conditions we also ran the two closest open-source systems through our harness (Table 4): Microsoft’s Agent Governance Toolkit [45] (agentmesh, commit 6b64456) and Bounded Agents [49]. Their runners (benchmarks/agentgovbench/competitors/) map each scenario onto the system’s own identity, delegation, policy, rate-limit and audit APIs; every decision and audit record is the system’s own, and features a system lacks are declared rather than emulated. The toolkit scores 45/48 on the official suite, 5/8 supplemental and 16/24 blind. Its losses trace to design choices its documentation states: rate-limit counters are keyed per agent identity, so fan-out multiplies a user’s allowance; a delegation that requests more than the parent holds is rejected rather than narrowed, leaving the child without an identity or chain; there is no fail-open mode (its ADR 0013); and its warn action blocks rather than flags. Because the first of these depends on how fan-out workers are modelled, we also report the configuration most favourable to it, in which workers act under the user’s own identity: 46/48, 6/8 and 17/24. Bounded Agents scores 36/48, 3/8 and 8/24: it has strong attenuation and evidence binding but no tiers, allow/deny policy layers, per-principal rate limits or fail-open, and several of its denials return before evidence is written, which the decision-coverage column shows. We installed the toolkit’s Python core without its Rust policy dependency, which the code path we use does not import; the full mapping, gaps and environment are in review/evidence/competitors-20260928/README.md.

Mechanism attribution. Table 5 removes one control at a time. Removing the decision audit costs 18 scenarios, the most of any control, because identity, provenance and audit categories all assert on audit records. All scope checks together account for 7 scenarios, tier rules for 5, and the delegation chain for 4. Two results expose defense in depth: removing only the control plane’s scope check changes a single scenario, because the gateway’s PDP re-checks scopes; and the one scenario it does change is the fail-open case, where the plane is the only layer that can supply the scope decision. In total 35 of 48 scenarios are attributed to at least one control; the other 13 expect benign calls to succeed and are satisfied by any system that does not over-deny.

Control removed	Official (48)		Supplemental (8)		Blind (24)	
	Score	Lost	Score	Lost	Score	Lost
Delegation attenuation [†]	48/48	0	6/8	2	21/24	3
Delegation chain in audit	44/48	4	7/8	1	20/24	4
Principal rate limiter	46/48	2	7/8	1	20/24	4
Locally declared fail mode	47/48	1	8/8	0	24/24	0
Honor fail-closed	46/48	2	8/8	0	22/24	2
Tenant membership	46/48	2	7/8	1	22/24	2
Deny-overrides resolution [†]	48/48	0	7/8	1	23/24	1
Per-call policy read	47/48	1	7/8	1	22/24	2
Decision audit	30/48	18	3/8	5	9/24	15
Plane scope check (PDP kept)	47/48	1	8/8	0	21/24	3
All scope checks	41/48	7	5/8	3	8/24	16
Tier rules	43/48	5	4/8	4	16/24	8
Reject anonymous principal	47/48	1	8/8	0	23/24	1
None (full system)	48/48	0	8/8	0	24/24	0

Table 5: Single-control ablations. Each row disables exactly one control after normal setup (benchmarks/agentgovbench/ablations.py).

“Lost” counts scenarios that fail. Supplemental: our 8 discriminating scenarios; Blind: 24 scenarios written without access to Agent Foundry’s code (§6.3). [†]Controls unobservable in the official suite but caught by the supplemental suite.

6.2 Do governance benchmarks measure enforcement?

Three observations answer RQ3. *First*, constant policies score well: a runner that denies every call and audits the denial scores 31/48, above the audit-only reference (29), and an allow-and-audit runner scores 25. Individually, 44 of the 48 scenarios are passed by at least one trivial runner; only 4 require both an allowed and a denied outcome. Aggregate scores still separate systems, but most single scenarios cannot distinguish enforcement from a constant. *Second*, two controls are unobservable: without attenuation, Agent Foundry still scores 48/48, because the only narrowing scenario (scope_inheritance.04) configures no subagent-tier rule, so a fail-closed default denies the call before narrowing matters; without deny-overrides it also scores 48/48, because no scenario contains conflicting incomparable rules. *Third*, published results show the same gap independently: 6 of the 8 published governed runs that declare “does not currently enforce task-scoped narrowing” nevertheless pass scope_inheritance.04.

Remedy. We wrote eight supplemental scenarios in the upstream format, each designed so that a specific control is necessary: narrowing with an allowed subagent tier; multi-hop narrowing where a grandchild re-requests a dropped scope; a user-wide allow against a tool-wide deny; one user ID in two tenants with different scopes; revocation affecting only its tier; concurrent rate limiting that must admit as well as deny; background-tier denial; and flagging. They are listed in Appendix A. Agent Foundry passes 8/8; vanilla, audit-only, deny-everything and allow-and-audit pass 0, 0, 0 and 0. Both previously unobservable controls now fail scenarios when removed (Table 5). On the official suite, Agent Foundry also satisfies all three anti-vacuity checks: decision coverage 48/48 (audit-only 47, vanilla 0), effect fidelity 48/48, and liveness 19/19 (deny-everything 0/19). The supplemental scenarios are authored by us and are reported separately from, never merged into, the official score.

6.3A blind, independently authored suite

The supplemental scenarios were written by the same party as the control plane, so they cannot answer whether the system was built to its tests. We therefore commissioned a separate suite from an author (an AI agent, instructed and audited for this) who was denied access to Agent Foundry’s source, runner, ablations, supplemental scenarios and this manuscript, and who worked

only from AgentGovBench’s own specification documents, loader, scorer and scenarios. The protocol required every scenario to assert both an admitted and a denied (or rate-capped) outcome, and targeted prop-erties the upstream suite under-tests: multi-hop narrowing on an allowed tier, conflicting rules at different specificities, one uid in two tenants, revocation and re-grant, interleaved rate limits across tiers, calls before, during and after an outage, and attribution of denied delegated calls. The author validated that an empty outcome, an all-allowed outcome and an all-denied outcome each fail all 24 scenarios, and recorded eleven judgment calls where the upstream semantics were ambiguous (benchmarks/agentgovbench/independent/README.md).

Run once, without any change to Agent Foundry, the suite gives Agent Foundry 24/24, against 0 for no governance, 1 for audit-only, and 0 and 0 for the two constant probes. Unlike the official suite, it observes 12 of the 13 controls (Table 5); the exception is locally declared fail mode, because its out-age scenarios read the policy before the outage begins. Two caveats limit this evidence. The author shares a model family with the system’s designer, so the suite is blind but not human-independent. And one judgment call—that a tool-wide deny overrides a user-wide allow—coincides with a de-sign decision of Agent Foundry; the author notes that a system following the precedence stated in an upstream scenario description would fail that one assertion.

ID	Defect observed before repair	Invariant	Regression
F01	Expired deadline, cancelled token or deny-all policy ignored: 2 model calls and 1 tool execution per engine	I1, I3	40 failing → 67 passing
F02	Approval resumed and executed; external policy engine consulted 0 times	I2	13 failing → 178 passing
F03	Viewer via native supervisor executed an admin-only tool; LangGraph denied	I1	(part of F01 suite)
F05	Two native workers on one store persisted [one, three] of three writes	I3	4 failing → 52 passing
F07	Knowledge passage with an injection marker and an e-mail reached the prompt	I6	2 failing → 333 passing
F08	Prompt cache ignored tool schemas: 2 requests, 1 provider call	I5	(part of F02 suite)
F09	Missing evaluation oracle counted as success; failures cost \$0	I7	17 failing → 70 passing

Table 6: Boundary defects found by probes and their regression suites (before → after, from the archived iteration logs). Tests run on both engines where applicable.

6.4 Boundary conformance across engines

Before the control plane existed, a review of the platform ran deterministic probes against both engines (review/evidence/reproduced-findings.json). Table 6 summarizes the defects and their repair; each repair was preceded by a regression suite that failed on the old code. Three findings came from the differential design: the native supervisor lost the caller’s identity where LangGraph denied (F03), and the request controls of F01 were dropped identically by both engines, which a single-engine test could not have distinguished from intended behavior without a specification.

The full suite at the campaign commit records 693 passing and 25 skipped tests (skips require un-available services), including 28 new tests for the control plane, the injection gate and the algebraic laws; the property tests check the semilattice laws on 400 random policy triples, attenuation along 300 random chains of up to six hops, and deny-monotonicity and limit tightening on 700 random rule sets. One run during development showed a single failing test that did not recur in 13 subsequent full runs; we could not identify it and report it rather than omit it.

6.5 Breadth: the remaining layers

Orchestration and multi-agent coordination overhead. Table 7 runs eight topologies with a scripted model, so it measures the engines themselves; both make identical model calls and incur identical synthetic cost. The native engine is 5.6–154× faster than the LangGraph engine at the median on the six multi-agent topologies, with the largest gaps on sequential multi-round topologies (blackboard, debate), where per-node graph overhead compounds. One contrary result is kept: with tool calls, the native p95 (19.22 ms) is worse than LangGraph’s (4.75 ms), plausibly because of thread-pool start-up in concurrent tool dispatch; we did not isolate the cause. Against model latency of several seconds per call, these differences matter for dense coordination and for cost of infrastructure, not for single-agent response time.

Topology	Native		LangGraph		Ratio
	p50	p95	p50	p95	
agent	0.11	0.16	1.99	2.78	18.1×
agent (tool-calling)	1.45	19.22	4.50	4.75	3.1×
supervisor	0.12	0.15	2.37	2.73	19.8×
swarm	0.22	0.25	2.78	3.05	12.6×
blackboard	0.53	0.56	70.95	75.57	133.9×
debate	0.35	0.38	53.81	54.68	153.7×
fanout	1.92	2.02	10.74	10.99	5.6×
dag	0.38	0.44	3.02	3.30	7.9×

Table 7: Engine overhead with a scripted model (medians of 5 runs; ms). Ratio is LangGraph p50 / native p50.

Thr.	Calls/s	p50	p95	p99	Limit	Audit
1	3,457	0.41	0.43	0.45	yes	yes
2	3,464	0.41	0.43	0.46	yes	yes
4	3,383	0.42	0.45	2.04	yes	yes
8	3,379	0.42	0.45	2.46	yes	yes
16	3,388	0.42	0.46	4.22	yes	yes
32	3,417	0.42	0.44	0.53	yes	yes

Proc.	Calls/s	Speedup	Eff.	Exact
1	3,583	1.00×	100%	yes
2	7,007	1.96×	98%	yes
4	13,578	3.79×	95%	yes
8	25,172	7.03×	88%	yes

Table 8: Scalability of admission. Left: one process, 32 principals in 4 tenants, 4800 calls, limit 100 of 150 attempts per principal. Right: principal-affine processes, 256 principals, limit 40 of 60.

Cost of governance. With a no-op tool, the median per-call cost is 175 μ s for the tool registry (allowlist and JSON-Schema validation dominate), 382 μ s for the governed gateway, 396 μ s for the control plane, and 400 μ s behind a three-hop delegation: the control plane adds 14 μ s to the gateway and each delegation chain adds 4 μ s. At the WorkBench median of roughly 5.7 s per task, governance is below 10^{-4} of end-to-end latency. Table 8 shows that one process sustains about 3,457 governed calls/s from 1 to 32 threads—CPython’s interpreter lock bounds parallel speedup—and that the limit and audit invariants hold exactly in all 10 repetitions at every level, as Theorem 3 predicts. Sharding by principal (Proposition 3) scales to 25,172 calls/s on 8 processes (7.0 $\times$, 88% efficiency) with exact limits. At roughly 3,583 calls/s per process, one million governed tool calls per minute needs about 6 shards; we did not measure network serving, distributed stores, or tenant-wide shared budgets.

Long context. We plant one fact at five depths in threads of 100 to 50,000 turns (2,865, 28,381, 285,415, 1,426,941 tokens) with three seeds, and require each strategy to produce context within a 2,000-token budget (Table 9). A recency window, the common default, keeps the fact only when it is recent (20% at the largest size); passing the full history keeps it but exceeds the budget by up to 732 $\times$. Foundry’s budgeted pipeline retains the fact in 100% of cells at every size with about 231 tokens of context, assembling 1,426,941 tokens of history in 68 ms. The default lexical retriever fails when the query is paraphrased (0%); swapping in the embedding store behind the same protocol restores retention to at least 87% at the sizes we could index (6.2 s to index 1,000 turns). With 20 planted injection passages that share the query’s keywords, the recency window admitted 5 into context across 6 runs and the Foundry pipeline admitted 0, while still returning the fact. This benchmark measures context *assembly*; whether a model then uses the fact is a separate question.

Security filter. On the public deepset/prompt-injections [17] test split (116 prompts, 60 attacks, revision 4f61ecb0), the default marker filter detects 0.0% of attacks (Wilson 95% [0.0, 6.0]) with no false positives; on the training split it detects 1.5%. Marker lists do not generalize to paraphrased or non-English attacks. We therefore added a dependency-free trained gate (injection_classifier, multinomial naive Bayes over word and character n-grams), fitted only on the 546-prompt training split with a fixed threshold. On the held-out test split it reaches recall 80.0% ([68.2, 88.2]), precision 94.1%, F1 0.865, false-positive rate 5.4%, and AUROC 0.874 ([0.809, 0.938]). It is now an optional detector on both the input guardrail and the context filter. Fine-tuned transformer classifiers can be expected to do better; the point is that the platform’s default statistical filter is no longer ineffective, and that filters remain a mitigation layered in front of least-authority tool access, not a boundary.

Strategy	100	1,000	10,000	50,000
Recency window	60/Y	20/Y	20/Y	20/Y
Full history	100/N	100/N	100/N	100/N
Foundry, lexical retriever	100/Y	100/Y	100/Y	100/Y
Foundry, lexical, paraphrased query	0/Y	0/Y	0/Y	0/Y
Foundry, embedding retriever	100/Y	100/Y	—	—
Foundry, embedding, paraphrased query	100/Y	87/Y	—	—

Table 9: Long-context assembly: retention (% of 15 depth $\times$ seed cells) / within budget (Y/N) by history length in turns. —: not run (embedding index cost).

Attack	Marker	NB	Both
Direct	0.0	24.8	24.8
Ignore previous	0.0	41.8	41.8
System message	0.0	27.0	27.0
InjecAgent	99.0	64.1	99.6
Important instructions	0.0	82.5	82.5

Suite	Marker	NB	Both
workspace	17.8 / 0.0	32.4 / 2.5	42.2 / 2.5
travel	20.0 / 0.0	62.7 / 2.4	64.5 / 2.4
banking	20.0 / 0.0	45.0 / 30.3	55.0 / 30.3
slack	20.0 / 0.0	85.0 / 37.3	85.0 / 37.3

Table 10: AgentDojo v1.2.2 injected tool outputs, model-free replay. Left: recall (%) per attack. Right: recall / false-positive rate (%) per suite.

Injected tool outputs: AgentDojo. Filters for tool outputs, not user prompts, are what indirect injection requires. Using AgentDojo [16] v1.2.2 without a model, we replay each of the 97 user tasks’ ground-truth tool calls in the clean environment (321 negative outputs) and in environments injected by five of AgentDojo’s registered attacks for every injection task (4,745 cases, 4,941 tool outputs that contain the injected goal). Table 10 reports detection. The marker filter catches the InjecAgent template and almost nothing else (overall recall 18.9%, false-positive rate 0.0%). The trained gate, fitted only on deepset’s training split

and therefore out of domain here, reaches 48.1% overall and is strongest on the important-instructions attack, but its false-positive rate is 14.3% and concentrated in the banking and Slack suites, whose ordinary tool outputs contain imperative text. Combined, recall is 54.9% at precision 98.3%. This transfer result is honest but weak: a deployer would need in-domain training data or a model-based detector, and filters remain a secondary layer. Utility and attack success with a model in the loop were not measured. **Hallucination scorer.** On HaluEval-QA [36] (10,000 pairs of a correct and a hallucinated answer with supporting knowledge, revision 12a85611), the grounding overlap behind reference_check_kpi separates correct from hallucinated answers with AUROC 0.906 ([0.901, 0.910]). Answer length alone does better, 0.971, because hallucinated answers are systematically longer. On the 384 pairs whose answer lengths differ by at most 20%, length falls to chance (0.513, [0.472, 0.554]) while grounding overlap keeps 0.917 ([0.896, 0.937]). The grounding signal is therefore real, and length-controlled evaluation is necessary to show it. The numeric fact-check scorer is uninformative here (AUROC 0.500), since few answers contain numeric claims.

Task execution: WorkBench. Table 11 reports the matched three-arm study. Differences from the reference loop are small and non-significant in every campaign; on the only untouched set the native engine solves 8 tasks the reference fails and fails 4 it solves (exact $p = 0.39$). We rescored all 24 published WorkBench model runs (16,560 predictions) and matched every published count; the best, Table 5, reaches 674/690 (97.68%) with its own harness. Our subsets use a smaller model and cannot be compared with it. The conclusion we draw is limited but useful: running tasks through Agent Foundry’s governed engines, with its default guards active, did not measurably cost task accuracy.

Campaign	R	N	LG	N vs R	LG vs R
Development (12)	10/12	12/12	10/12	2/0, $p=0.50$	2/2, $p=1.00$
First 60 (inspected)	50/60	49/60	48/60	3/4, $p=1.00$	4/6, $p=0.75$
Guidance, same 60	47/60	49/60	49/60	4/2, $p=0.69$	5/3, $p=0.73$
Fresh 60 (untouched)	45/59	50/60	49/60	8/4, $p=0.39$	6/2, $p=0.29$

Table 11: WorkBench, matched arms (R reference loop, N native engine, LG LangGraph engine), one repetition each, gpt-5.4-mini. Discordant pairs are Foundry-only/reference-only successes with exact McNemar p .

Dimension	Evidence	Verdict
Governance	48/48 (10/10 runs) vs. best published 46; measured AGT 46, Bounded Agents 36; blind 24/24 vs. 17 and 8	SOTA on AgentGovBench
Measurement validity	2 unobservable controls found and made observable; anti-vacuity checks; blind suite 24/24	New for enforcement benchmarks
Coordination overhead	5.6–154× lower median than LangGraph on 6 topologies	Better than LangGraph
Governance cost, scale	+14 μ s/call; exact limits to 32 threads; 7.0× on 8 shards	Adequate; network scale untested
Long-context assembly	100% vs. 20% (recency) within budget	Better than recency default
Injection filter	recall 0.0% → 80.0% (held out)	Repaired; below model-based detectors
Hallucination scorer	length-matched AUROC 0.917	Informative, lexical
Task execution	no significant difference from reference; best published 97.68%	Not SOTA
Multi-agent quality, planning	not measured (requires model-driven benchmarks)	Unmeasured

Table 12: 360-degree scorecard. “SOTA” is used only where a published comparator was run under the same scorer.

6.6 Summary scorecard

Table 12 collects the verdicts.

VII.DISCUSSION

What any agent framework should adopt. The results suggest a short checklist that is independent of Agent Foundry’s code. (1) Establish controls once, at entry, as an execution scope that every boundary—threads, generators, resume, fan-out, exports—re-enters, and compose nested scopes by a meet. (2) Never persist live handles; re-derive controls on resume. (3) Decide every effect at one deny-first admission point, and evaluate rate limits last so denials do not consume capacity. (4) Key limits by principal, not by agent or thread, so that fan-out cannot multiply a quota, and shard by that key. (5) Make the fail mode local configuration (Proposition 2). (6) Emit exactly one audit record per decision, including denials and outages. (7) Treat unmeasured as distinct from passed, and report governance scores together with ablations or trivial-runner baselines; a score that a deny-everything runner nearly matches says little by itself.

Why governance, and not task accuracy, is where the architecture shows. Task benchmarks such as WorkBench are dominated by the model: published full-set runs range from 406 of 690 for the model we used to 674 of 690 for the strongest, a

spread that dwarfs the differences among our arms. A governed platform's value is that controls hold when the model, the topology or the de-ployment changes, which is exactly what task accuracy does not measure. Conversely, governance benchmarks are cheap to run and deterministic, which makes them easy to overfit; mechanism attribution and discriminating scenarios are the counterweight.

7.1 Threats to validity

Benchmark-driven development. The control plane was written during this study, after reading AgentGovBench's scenarios. A system built to a test suite can pass it without generalizing. We mitigate this in three ways, none of which removes the threat: every allow/deny decision is made by library code that has its own unit and property tests independent of the benchmark; ablations show that each passing category depends on a mechanism rather than on scenario-specific logic; and eight supplemental scenarios probe behavior the official suite does not. The supplemental scenarios are ours; the blind suite (§6.3) was written without access to our code, but by an AI author from the same model family, so a suite written by independent humans would be stronger still.

Harness choices. The virtual clock and threaded fan-out differ from the upstream runners, and both are disclosed. Threaded fan-out is stricter. The virtual clock matters only for the one recovery scenario, where the upstream reference runner sleeps through the same outage.

Comparators. Published Agent Gov Bench rows were produced by their authors against a live gate-way, not rerun by us; their declared-unsupported scenarios are counted as published. The runners for the Agent Governance Toolkit and Bounded Agents were written by us; a mapping by their authors might score higher, and we report the most favourable of the two toolkit configurations we tried. The trivial runners are ours.

Scope of the non-governance benchmarks. The long-context benchmark is synthetic and measures assembly, not use by a model; the injection data set is small (116 test prompts) and partly non-English; HaluEval-QA has the length artifact we document; the scaling results are from one machine; and the WorkBench study uses one model, one repetition and subsets of 12 and 60 tasks.

7.2 Limitations

We did not measure the utility cost of attenuation, which MasDrift [71] and Bounded Agents [49] show can be substantial; AgentGovBench has no model and therefore cannot. The audit log is not tamper-evident, approvals are not bound to action digests or consumed once, and there is no information-flow tracking; the Agent Governance Toolkit, Bounded Agents and CaMeL/FIDES are ahead on each. Process-local scopes are not an isolation boundary against malicious Python tools; the platform's in-process sandbox is for trusted code only. Cooperative cancellation does not stop a tool already running. Distributed budgets, fenced ownership of shared state, durable reconciliation of external effects (no exactly-once guarantee), authenticated approval resume for exported tools, and opaque CrewAI internals remain open, as does multi-agent planning quality, which we did not evaluate. AgentDojo, τ^2 -bench and MultiAgentBench require model-driven runs that we did not perform. The trained injection gate is a lexical model and should be replaced by or combined with a model-based detector where available.

Broader impact. Explicit, testable controls can make it safer for small teams to deploy agents that take actions. The same results could be over-read: a perfect governance score does not certify a deployment, and filters do not make untrusted content safe. We report the conditions under which each result holds and the mechanisms it depends on, so that readers can judge transfer to their setting.

VIII. CONCLUSION

Controls attached to a request must survive every boundary a platform creates, and the way to make them survive is architectural: represent them as a composable execution scope and decide every effect at one deny-first admission point. We formalized this as the Governed Execution Contract, proved its central properties, implemented it in Agent Foundry, and showed a perfect, repeatable AgentGovBench score whose passing scenarios are attributed to specific mechanisms. The same analysis shows that governance benchmarks need ablations, trivial baselines and discriminating scenarios before their scores can be read as evidence of enforcement, and we supply those. Across the other layers the platform is competitive or better than common defaults, and where it was not— injection filtering and paraphrase retrieval—the benchmarks told us what to fix.

Acknowledgments and Disclosure of Funding

AI assistance (Anthropic Claude) was used for implementation, literature search and citation check-ing, experiment orchestration and drafting. All experiments were executed by scripts in the repository, and every reported number is regenerated from raw evidence by build_evidence.py. The author is responsible for the content. No external funding was received.

REFERENCES

1. Agentic Control Plane. AgentGovBench, 2026. Source revision e0ce93ae1753. URL: <https://github.com/agentic-control-plane/agentgovbench>.
2. Agentic Control Plane. Agentic Control Plane: Governance model and AgentGovBench score-card (v0.2.6). <https://agenticcontrolplane.com/benchmark>, 2026. Accessed 2026-09-28.
3. Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -bench: Evaluating conversational agents in a dual-control environment. arXiv preprint arXiv:2506.07982, 2025. arXiv version inspected: v1. URL: <https://arxiv.org/abs/2506.07982>, arXiv: 2506.07982.
4. Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient detection of vacuity in temporal model checking. Formal Methods in System Design, 18(2):141–163, 2001. doi: 10.1023/A:1008779610539.
5. Roy Betser, Amit Giloni, Shamik Bose, Sindhu Padakandla, Chiara Picardi, Lidor Erez, and Roman Vainshtein. Agentrim: Tool risk mitigation for agentic ai. arXiv preprint arXiv:2601.12449, 2026. Findings of EMNLP 2026 (per arXiv comment); arXiv version in-

- spected: v2. URL: <https://arxiv.org/abs/2601.12449>, arXiv:2601.12449.
6. Luca Beurer-Kellner, Beat Buesser, Ana-Maria Crețu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naef, Ezinwanne Ozoani, Andrew Paverd, Florian Tramèr, and Václav Volhejn. Design patterns for securing llm agents against prompt injections. arXiv preprint arXiv:2506.08837, 2025. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2506.08837>, arXiv:2506.08837.
7. Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Ti-wari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2503.13657>, arXiv:2503.13657.
8. Edward Y. Chang and Longling Geng. Mnemosyne: Agentic transaction processing for vali-dating and repairing ai-generated workflows. arXiv preprint arXiv:2607.00269, 2026. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2607.00269>, arXiv:2607.00269.
9. Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding agents via verifiable safety policy reasoning, 2025. URL: <https://arxiv.org/abs/2503.22738>, arXiv:2503.22738.
10. Moonwon Choi, Seokho Jeong, and Seunggeun Lee. ResidualAuth: What authorization state must language agents preserve under revocable delegation?, 2026. URL: <https://arxiv.org/abs/2609.08062>, arXiv:2609.08062.
11. 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, 2025. URL: <https://arxiv.org/abs/2505.23643>, arXiv:2505.23643.
12. CrewAI. CrewAI, 2026. Source revision 4ed2abc7bbf5. URL: <https://github.com/crewAIInc/crewAI>.
13. Alejandro Cuadron, Pengfei Yu, Yang Liu, and Arpit Gupta. Saber: Small actions, big errors – safeguarding mutating steps in llm agents. arXiv preprint arXiv:2512.07850, 2025. arXiv version inspected: v1. URL: <https://arxiv.org/abs/2512.07850>, arXiv:2512.07850.
14. Panduranga Sai Varma Dantuluri and Jyotirmoy Sundi. Delegation without trust: An empirical gap analysis of identity, authorization, and runtime governance in multi-agent LLM systems, 2026. URL: <https://arxiv.org/abs/2609.00267>, arXiv:2609.00267.
15. Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. arXiv preprint arXiv:2503.18813, 2025. arXiv version inspected: v2. URL: <https://arxiv.org/abs/2503.18813>, arXiv:2503.18813.
16. Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2406.13352>, arXiv:2406.13352.
17. deepset. deepset/prompt-injections. Hugging Face dataset, 2023. Revision 4f61ecb. URL: <https://huggingface.co/datasets/deepset/prompt-injections>.
18. Runzhi Deng, Yiming Zhong, Fang Zhao, and Pan Zhou. Do not restart: Residual completion for stateful agent handoffs. arXiv preprint arXiv:2609.13800, 2026. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2609.13800>, arXiv:2609.13800.
19. Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed compu-tations. *Communications of the ACM*, 9(3):143–155, 1966. doi:10.1145/365230.365252.
20. Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 11733–11763, 2024. arXiv version inspected: v1. URL: <https://arxiv.org/abs/2305.14325>, arXiv:2305.14325.
21. Google. Agent Development Kit (ADK) for python. <https://github.com/google/adk-python>, 2026. Commit 044a1ec, accessed 2026-09-28.
22. Guardrails AI. Guardrails: Adding guardrails to large language models. <https://github.com/guardrails-ai/guardrails>, 2026. Commit 06d0ff2, accessed 2026-09-28.
23. Norm Hardy. The confused deputy: (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38, 1988. doi:10.1145/54289.871709.
24. Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiaowu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv version inspected: v7. URL: <https://arxiv.org/abs/2308.00352>, arXiv:2308.00352.
25. Amjad Ibrahim and Yong Li. Overlaying governance: A compositional authorization frame-work for delegation and scope in agentic AI, 2026. URL: <https://arxiv.org/abs/2606.03518>, arXiv:2606.03518.
26. Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabisa. Llama Guard: LLM-based input-output safeguard for human-AI conversations, 2023. URL: <https://arxiv.org/abs/2312.06674>, arXiv:2312.06674.
27. Invariant Labs. Invariant Guardrails: Contextual guardrails for securing agent systems. <https://github.com/invariantlabs-ai/invariant>, 2026. Commit 2340fe2, accessed 2026-09-28; Invariant Labs acquired by Snyk, June 2025.
28. Zimo Ji, Daoyuan Wu, Wenyan Jiang, Pingchuan Ma, Zongjie Li, Yudong Gao, Shuai Wang, and Yingjiu Li. Taming various privilege escalation in LLM-based agent systems: A mandatory access control framework, 2026. URL: <https://arxiv.org/abs/2601.11893>, arXiv:2601.11893.
29. Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011. doi:10.1109/TSE.2010.62.
30. Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. arXiv preprint arXiv:2407.01502, 2024. arXiv version inspected: v1. URL: <https://arxiv.org/abs/2407.01502>, arXiv:2407.01502.
31. Anuj Kaul, Qianlong Lan, and Pranay Gupta. Behavioral governance for autonomous AI agents: The AgentBound framework, 2026. URL: <https://arxiv.org/abs/2606.30970>, arXiv:2606.30970.
32. LangChain. Deep Agents, 2026. Source revision 0c756c4f8037. URL: <https://github.com/langchain-ai/deepagents>.
33. LangChain. LangGraph, 2026. Source revision 7daa3ab49d67. URL: <https://github.com/langchain-ai/langgraph>.
34. Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv version

- inspected: v2. URL: <https://arxiv.org/abs/2303.17760>, arXiv:2303.17760.
35. Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents, 2025. NeurIPS 2025. URL: <https://arxiv.org/abs/2506.12104>, arXiv:2506.12104.
36. Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. HaluEval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. URL: <https://arxiv.org/abs/2305.11747>.
37. Lifei Liu and Haoran Yu. Attesting outputs and delegation ancestry in multi-agent AI systems, 2026. URL: <https://arxiv.org/abs/2608.30387>, arXiv:2608.30387.
38. Ting Liu. From tool connection to execution control: Benchmarking security invariants in MCP-style agent runtimes, 2026. URL: <https://arxiv.org/abs/2606.29073>, arXiv: 2606.29073.
39. Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2308.03688>, arXiv:2308.03688.
40. Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegr-eff, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv version inspected: v2. URL: <https://arxiv.org/abs/2303.17651>, arXiv:2303.17651.
41. William M. McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998.
42. David Mellafe Zuvic. Capability gates are not authorization: Confused-deputy failures in LLM agent frameworks, 2026. URL: <https://arxiv.org/abs/2606.28679>, arXiv:2606. 28679.
43. Meta. Llama Prompt Guard 2 model card (86m and 22m). <https://huggingface.co/meta-llama/Llama-Prompt-Guard-2-86M>, 2025. Released 2025-04-05.
44. Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv version inspected: v1. URL: <https://arxiv.org/abs/2311.12983>, arXiv:2311.12983.
45. Microsoft. Agent Governance Toolkit, 2026. Source revision 6b644564d112. URL: <https://github.com/microsoft/agent-governance-toolkit>.
46. Microsoft. Microsoft Agent Framework, 2026. Source revision 6f1522a50b66. URL: <https://github.com/microsoft/agent-framework>.
47. Mark Samuel Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, Baltimore, MD, USA, 2006. URL: <http://www.erights.org/talks/thesis/>.
48. Bardia Mohammadi and Laurent Bindschaedler. The irreversibility budget: Fleet-level risk accounting and admission control for agent operating systems, 2026. URL: <https://arxiv.org/abs/2609.00275>, arXiv:2609.00275.
49. Xabier Muruaga. Bounded agents: Delegation security for multi-agent AI systems, 2026. URL: <https://arxiv.org/abs/2608.15888>, arXiv:2608.15888.
50. Yi Nian, Aojie Yuan, Haiyue Zhang, Jiate Li, Li Li, Xiyang Hu, Hua Wei, Xiongye Xiao, Chaowei Xiao, and Yue Zhao. Auditable agents, 2026. URL: <https://arxiv.org/abs/2604.05485>, arXiv:2604.05485.
51. OpenAI. OpenAI Agents SDK (openai-agents-python). <https://github.com/openai/openai-agents-python>, 2026. Commit 08e5c43, accessed 2026-09-28.
52. Yuxiang Peng and Xiaodi Wu. Stateful governance for concurrent agentic systems, 2026. URL: <https://arxiv.org/abs/2608.02764>, arXiv:2608.02764.
53. Pydantic. Pydantic AI, 2026. Source revision 69eb81bde23c. URL: <https://github.com/pydantic/pydantic-ai>.
54. Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails, 2023. EMNLP 2023 System Demonstrations. URL: <https://arxiv.org/abs/2310.10501>, arXiv:2310.10501.
55. Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi:10.1109/PROC.1975.9939.
56. Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2025. URL: <https://arxiv.org/abs/2504.11703>, arXiv:2504.11703.
57. Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv version inspected: v4. URL: <https://arxiv.org/abs/2303.11366>, arXiv:2303.11366.
58. Oleg Solozobov. DEMM-Bench: A cross-regime benchmark for agent-runtime governance-evidence sufficiency, 2026. URL: <https://arxiv.org/abs/2606.20634>, arXiv:2606. 20634.
59. Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, Cedric Deslandes Whitney, Dazza Greenwood, Alan Chan, and Alex Pentland. Authenticated delegation and authorized AI agents, 2025. URL: <https://arxiv.org/abs/2501.09674>, arXiv:2501.09674.
60. Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. Workbench: a benchmark dataset for agents in a realistic workplace setting. arXiv preprint arXiv:2405.00823, 2024. arXiv version inspected: v2. URL: <https://arxiv.org/abs/2405.00823>, arXiv:2405.00823.
61. Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2309.02427>, arXiv:2309.02427.
62. Krti Tallam. A five-plane reference architecture for runtime governance of production AI agents, 2026. URL: <https://arxiv.org/abs/2606.12320>, arXiv:2606.12320.
63. Lillian Tsai and Eugene Bagdasarian. Contextual agent security: A policy for every purpose, 2025. HotOS 2025. URL: <https://arxiv.org/abs/2501.17070>, arXiv:2501.17070.
64. Rasmus Moorits Veski, Rachid Guerraoui, and David Froelicher. CAPMAS: Capability-based delegation of privileges in multi-agent systems, 2026. URL: <https://arxiv.org/abs/2609.06500>, arXiv:2609.06500.
65. Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 2938–2950. ACM, 2026. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2503.18666>, arXiv:2503.18666.

66. Haoyu Wang, Christopher M. Poskitt, Jiali Wei, and Jun Sun. Probguard: Proactive runtime monitoring for llm agent safety via probabilistic prediction. arXiv preprint arXiv:2508.00500, 2025. v1 titled “Pro2Guard: Proactive Runtime Enforcement of LLM Agent Safety via Probabilistic Model Checking”; retitled ProGuard by v4; accepted at ASE 2026 per arXiv comment; arXiv version inspected: v4. URL: <https://arxiv.org/abs/2508.00500>, arXiv:2508.00500.
67. Youting Wang, Xiao Han, Dingyan Shang, Yuan Tang, and Bowen Liu. Safety, or just capability? A validity audit of agent-safety benchmarks, 2026. URL: <https://arxiv.org/abs/2607.28685>, arXiv:2607.28685.
68. Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024. arXiv version inspected: v2. URL: <https://arxiv.org/abs/2308.08155>, arXiv:2308.08155.
69. Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning, 2025. ICML 2025. URL: <https://arxiv.org/abs/2406.09187>, arXiv:2406.09187.
70. Jinghan Xu, Longze Fan, Zeyuan Wang, Xinjin Li, and Hankai Liu. Beyond single-use tokens: Durable authorization state for replay-resistant LLM agent actions, 2026. URL: <https://arxiv.org/abs/2608.01710>, arXiv:2608.01710.
71. Zhuoning Xu, Xiucheng Zhang, Hanjun Luo, Yingbin Jin, Yinpeng Dong, and Hanan Salam. MasDrift: Benchmarking authorization preservation across multi-agent architectures, 2026. URL: <https://arxiv.org/abs/2608.07556>, arXiv:2608.07556.
72. Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv version inspected: v1. URL: <https://arxiv.org/abs/2406.12045>, arXiv:2406.12045.
73. Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2210.03629>, arXiv:2210.03629.
74. Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. arXiv version inspected: v3. URL: <https://arxiv.org/abs/2403.02691>, arXiv:2403.02691.
75. Kaiyuan Zhang, Yuke Peng, Ke Jiang, and Yinqian Zhang. Actgov: Governing llm agent actions via policy-constrained validation. arXiv preprint arXiv:2609.24446, 2026. arXiv version inspected: v2. URL: <https://arxiv.org/abs/2609.24446>, arXiv:2609.24446.
76. Jinhao Zhu, Kevin Tseng, Gil Vernik, Xiao Huang, Shishir G. Patil, Vivian Fang, and Raluca Ada Popa. MiniScope: A least privilege framework for authorizing tool calling agents, 2025. URL: <https://arxiv.org/abs/2512.11147>, arXiv:2512.11147.
77. Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwetmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices for building rigorous agentic benchmarks, 2025. URL: <https://arxiv.org/abs/2507.02825>, arXiv:2507.02825.

A Supplemental discriminating scenarios

Each scenario uses only official assertion kinds and the upstream YAML format; files are in `benchmarks/agentgovbench/supplemental/scenarios/discrimination/`. The last column names the control whose removal fails the scenario in the second campaign.

#	Setup and expectation	Control made observable
01	Bob (admin) delegates only <code>tools.read</code> to a summarizer on an <i>allowed</i> subagent tier; the summarizer’s read succeeds and its <code>grant_permission</code> is denied	attenuation
02	Orchestrator $\rightarrow$ specialist <code>{tools.read}</code> $\rightarrow$ worker <code>{tools.read,email.read}</code> ; the worker’s <code>read_email</code> is denied, its file read allowed, and the chain recorded	attenuation, chain
03	User-wide allow and tool-wide deny for <code>send_email</code> ; the call is denied, an unrelated read allowed	deny-overrides
04	The uid <code>sam</code> exists in two tenants with different scopes; <code>grant_permission</code> is denied in the first, allowed in the second, and the denial is audited in the right tenant	tenant membership
05	A tool-level revocation denies the next interactive call but not the subagent tier	per-call policy read, tier rules
06	Ten concurrent workers make 200 calls under a limit of 60; at most 60 admitted, and both an allow and a deny are audited	rate limiter, audit
07	A background-tier read is denied while an interactive read is allowed, both attributed	tier rules
08	A <code>flag</code> rule admits the call and records <code>decision=flag</code>	tier rules, audit

B. Ablation definitions

Ablations patch runner-owned objects after setup (`benchmarks/agentgovbench/ablations.py`); the production control plane has no ablation flags. *Delegation attenuation*: delegated scopes ignored. *Delegation chain*: chain omitted from audit. *Principal rate limiter*: limiter always admits. *Locally declared fail mode*: fail mode learned only from successful reads (the original design). *Honor fail-closed*: outages always fail open. *Tenant membership*: a principal known in any tenant is accepted in every tenant. *Deny-overrides*: the first defined rule in the order user-tool, user, tool, default wins. *Per-call policy read*: a policy snapshot from setup is used. *Decision audit*: no audit records. *Plane scope check*: the control plane’s required-scope check is removed while the gateway PDP keeps its own. *All scope checks*: both removed. *Tier rules*: every tier resolves to allow. *Anony-*

mous principal: an empty uid is replaced by a member principal. The two probes, *deny-everything* and *allow-and-audit*, bypass enforcement entirely but still write audit records through the plane’s recorder.

C. Anti-vacuity checks

For each official scenario the harness records the number of attempted calls, audited decisions, allowed calls and, for runners that expose them, executed tool effects. *Decision coverage* holds if attempts > 0 and decisions = attempts. *Effect fidelity* holds if executed effects = allowed calls, so an allowed decision that never ran the tool, or an effect without an allowed decision, is detected. *Liveness* applies to scenarios whose expectations include *tool_allowed* or *rate_limited_count* and holds if at least one call was admitted; it rejects a deny-everything system on exactly the scenarios whose upper-bound assertions it otherwise passes. These checks are reported beside, not inside, the official score.

D. Evidence and reproduction

All commands run from the repository root with Python 3.11 and the recorded dependencies.

```
git clone https://github.com/agentive-control-plane/agentgovbench /tmp/agb git -C /tmp/agb checkout
e0ce93ae175376d7847c69a64d0c36bdfa6ca717
pip install -e "[all,test]" pyyaml click hypothesis pyarrow chromadb python
benchmarks/run_paper_campaign.py --agentgovbench /tmp/agb \
--out review/evidence/paper-campaign-

```

Costs. All experiments in this paper except WorkBench make no model calls. The WorkBench study made 1,855 provider requests under a \$25 ledger; usage-based upper estimates plus full reservations for requests with unknown usage total \$16.72; these are not invoices.

E. Scenario-level vacuity

Of the 48 official scenarios, 44 are passed by at least one of the four trivial runners (no governance, audit-only, deny-everything, allow-and-audit) and 4 by none. The non-vacuous scenarios are those that require an allowed and a denied outcome for different principals, tools or times within one scenario (for example, a user override that allows one principal while denying another). A benchmark that wants per-scenario diagnostic value, rather than only an aggregate, benefits from pairing each deny expectation with an allow expectation that a constant policy cannot satisfy.

F. Per-invariant comparison

Y: provided; P: partial or opt-in; N: absent or contrary; –: out of scope; “opp.”: the work argues for the opposite design. Frameworks were inspected in source at pinned revisions; papers from their method sections. FM: fail mode known locally per tenant; Aud: one audited decision per call; Shd: coordination-free per-principal partitioning. This is a structured reading, not an execution of each system.

System	I1	I2	I3	I4	I5	I6	I7	FM	Aud	Shd
Agent Foundry	Y	<i>Ya</i>	Y	Y	Y	<i>pb</i>	Y	Y	Y	<i>pc</i>
MS Agent Governance Toolkit	Y	Y	opp.	P	–	P	N	P	<i>Yd</i>	P
ACP gateway	P	P	–	P	–	P	N	N	Y	–
MS Agent Framework	N	P	P	N	N	P	–	Y	P	N
LangGraph	N	N	P	N	N	N	–	N	P	N
Deep Agents	N	N	P	N	–	N	–	N	N	N
Pydantic AI	N	P	P	P	–	N	P	N	P	N
CrewAI	N	N	P	P	N	P	–	N	N	N
OpenAI Agents SDK	N	N	P	P	–	P	–	N	P	N
Google ADK	N	N	P	N	P	P	P	N	P	N
NeMo Guardrails	–	P	–	–	P	Y	N	P	P	N
Invariant Guardrails	N	Y	–	–	–	Y	N	N	P	N
Mnemosyne / ATP	N	P	P	N	–	N	P	Y	Y	P
CaMeL; FIDES	N	P	–	N	N	<i>pe</i>	–	P	N	N

Progent	P	Y	–	N	N	N	–	Y	N	N
ActGov	P	Y	–	N	N	P	N	P	P	N
Bounded Agents	Y	P	N	<i>pg</i>	N	N	P	Y	Y	N
CAPMAS; Overlaying Gov.	Y	P	N	N	N	N	–	P	P	N
Irreversibility Budget	N	–	N	Y	N	N	–	Y	P	P
MasuGate	N	P	N	<i>Yf</i>	N	N	P	Y	Y	N
CapLease; ResidualAuth	P	P	opp.	P	N	N	P	Y	P	N
Liu, MCP invariants	N	P	N	N	N	P	–	P	Y	N

*a*Specificity first: a user-and-tool rule wins; deny-overrides combines incomparable rules. *b*Same detectors for all content; retrieved passages dropped and redacted, tool results labelled. *c*Router demonstrated in a benchmark, not a library component. *d*Hash-chained, stronger than ours. *e*Information-flow control, stronger than ours. *f*Concurrency-safe across effects, stronger than ours. *g*Per delegation chain (consumed budget inherited as a floor).