Before AI Agents Act: Policy-Gated Data Pipelines for Secure Enterprise Agentic AI

Ankush Ramprakash Gautam
Vice President, Data Engineering at Fidelity Investments

ABSTRACT

Enterprise AI agents convert retrieved data into consequential tool calls, yet conventional data quality validation does not establish authority to mutate a resource. This paper specifies a policy-gated authorization boundary that checks schema, freshness, source provenance, tenant and tool scope, principal status, replay state, and risk-dependent approval before an external action. A deterministic Python prototype is compared against no pre-action checks and schema/freshness screening on 7,200 labeled synthetic proposals; seven single-control ablations and a separately generated 2,400-case stress set examine boundary and compound violations. The full gate returns 1,200 ALLOW, 5,400 DENY, and 600 REVIEW decisions in the primary suite, with no authorization that violates the stipulated policy; the quality-only baseline incorrectly allows 4,800 proposals. On the stress set, the full gate retains all 600 valid proposals and routes 300 approval-dependent cases to review, whereas the quality-only baseline incorrectly allows 1,500. Counterfactual removals expose the contribution of individual controls, while visual comparisons make the tradeoffs explicit. These findings establish consistency with authored synthetic policy rules, not resistance to adaptive attackers or end-to-end production safety. The proposed design separates trusted data assertions, independent policy decisions, and enforcement at the write API, and identifies the experiments needed for external security validation.

General Terms

Security, Data Engineering, Access Control, Artificial Intelligence, Distributed Systems

Keywords

Agentic AI; Data Engineering; Policy-Gated Pipelines; Data Provenance; Tool Authorization; AI Agent Security; Enterprise Governance

1. INTRODUCTION

Autonomous enterprise agents increasingly retrieve business records, interpret retrieved material, and invoke applications through tools. Architectures such as ReAct and Toolformer established the practical role of language-model-driven tool use [1] [2]. Yet an otherwise syntactically valid request can produce an impermissible effect: a case management agent may update an account using a record from another tenant, approve a stale exception, or invoke a payment tool after reading manipulated source content. Unlike answer generation errors, such failures mutate enterprise state and may be difficult to reverse.

Indirect prompt injection illustrates how external content can be promoted into unintended instructions [3]. AgentDojo demonstrates the importance of evaluating agent tool behavior in realistic adversarial environments [4]. OWASP's agentic risk guidance likewise emphasizes the operational consequences of tool and privilege misuse [5]. The resulting engineering problem is not merely whether a retrieved record is correct, but whether an identified principal may perform a particular action, on a particular resource, using that record, at the moment the action executes.

Earlier Data Engineering work on clean context pipelines treats inference context as a governed data product [6], while evidence on data quality and LLM hallucinations addresses reliability of generated content [7]. Those concerns remain relevant, but they do not themselves constitute action authorization. This study moves the enforcement point from context preparation to the boundary between agent intent and enterprise action.

The research question is: can a policy-gated Data Engineering layer distinguish allowable, rejectable, and approval-requiring actions using source, freshness, identity, scope, and approval metadata without prohibitive local processing overhead? This paper contributes (i) a concrete action-bound policy decision model; (ii) a reference architecture with explicit trust boundaries and audit binding; and (iii) a reproducible synthetic microbenchmark comparing no validation, ordinary record quality screening, and pre-execution authorization. The evaluation is deliberately bounded: it measures a reference policy implementation, not an LLM model, a cloud deployment, or field attack resistance.

2. RELATED WORK AND RESEARCH GAP

Agentic Data Engineering literature is beginning to integrate data contracts, expectations, provenance, and governance. Cravero et al. propose a broad framework for quality of agentic data use, with a small prototype and benchmark, while appropriately avoiding claims of generalized protection [8]. The framework supplies a useful conceptual foundation. The present work asks a narrower operational question: what must a transaction facing authorization gate verify immediately before a tool receives an actionable request, and what does an executable comparison reveal about the value of that gate?

The NIST Generative AI Profile situates these risks within systematic risk management [9]. Zero trust architecture likewise motivates checking resource access at each interaction rather than treating an agent session or network location as inherently trusted [10]. The W3C PROV Ontology (PROV-O) provides a standardized OWL 2 vocabulary for expressing relationships among entities, activities and agents [11]. It supports interoperable provenance representation, but neither authenticates provenance assertions nor grants permission to invoke a tool. Open Policy Agent is an example of an independently operated policy decision point; the present Python benchmark does not execute Rego policies [12].

There is a related distinction between context relevance and permission to act. Long context experiments have shown that merely supplying additional information does not guarantee that a language model uses the correct evidence [13]. This paper therefore does not ask a language model to self-certify the legitimacy of a tool invocation. It specifies deterministic predicates evaluated outside the model, with the execution

gateway as the required enforcement point.

3. THREAT MODEL AND DESIGN REQUIREMENTS

3.1 Assets, actors and trust boundaries

The protected assets are mutable enterprise records and externally visible side effects, including case updates, workflow transitions, and API operations. A principal is the workload or end user identity on whose behalf an agent acts. The model may suggest an action but is not a security principal, policy administrator, or source of trusted authorization metadata. Tool descriptions, retrieved documents, generated rationales, and natural language instructions are untrusted unless independently authenticated.

The design assumes that source provenance assertions, identity attributes, authorizations and approval attestations are supplied by independently trusted enterprise services. A malicious or mistaken agent can alter requested tool, resource, argument, or source reference, but cannot directly modify those trusted assertions. This is an explicit security assumption rather than a property established by the prototype. Backend APIs must reject direct calls that bypass the gate.

3.2 Concrete action level invariants

A1 Identity binding: a proposed action is evaluated against an authenticated and active principal. A2 Source admissibility: evidence has an accepted source identity and freshness bound. A3 Resource isolation: tenant and resource scope are authorized for the principal. A4 Tool minimization: the action's API tool is explicitly allowlisted. A5 Human oversight: designated high-risk actions require a valid approval attestation. A6 Replay resistance: an action nonce must be unused, and time-sensitive permissions must be checked at execution. A7 Accountability: each decision is paired with a reproducible evidence record and policy revision.

The gate fails closed when necessary metadata is missing or unverifiable. It returns DENY when a hard constraint fails, REVIEW when all hard constraints pass but a valid high-risk approval is absent, and ALLOW only when the full predicate succeeds. REVIEW is not an authorization to execute; it is an instruction to place a request in an approval workflow. An expired approval, by contrast, is DENY.

3.3 Threat cases and scope of the defense

A threat case is a candidate action with an identified security property at stake, not merely an anomalous model response. Four representative paths are considered. A retrieved ticket can contain a command to invoke an unauthorized transfer API; a record can be technically well formed but originate from an unapproved source; a tenant A request can target a tenant B resource; and a valid looking high-risk update can lack current approval. The first path illustrates indirect prompt injection risk; the prototype does not inspect, classify, or neutralize such language. Protection instead requires a separate execution policy that rejects the unauthorized tool or fails a source-attestation check. These scenarios are mapped to independently supplied metadata predicates, not to model confidence scores.

The protected boundary begins after an agent proposes an action and ends when the actual side-effecting API either accepts or rejects that operation. Out of scope are initial user authentication, correctness of business semantics, compromised identity providers, leaked execution credentials, forged trusted attestations, and a backend exposed through an unprotected alternate route. In each of these cases, a correct local policy decision alone would be insufficient. The prototype models one proposed action at a time; it cannot enforce a multi step spending or transaction quota without shared transactional state.

3.4 Data plane ownership and control plane ownership

The data plane owns typed source records, record version, observation timestamp, and trusted producer identity. The security control plane owns the effective principal, tenant membership, tool grants, approval state, and revocation epoch. Their attributes must be combined using an immutable action identifier at the execution gateway. An LLM generated explanation is retained as untrusted context rather than permitted to alter either set of authoritative facts. This separation is the principal difference between improving the quality of context and enforcing authorization for a state change.

4. PROPOSED POLICY-GATED ARCHITECTURE

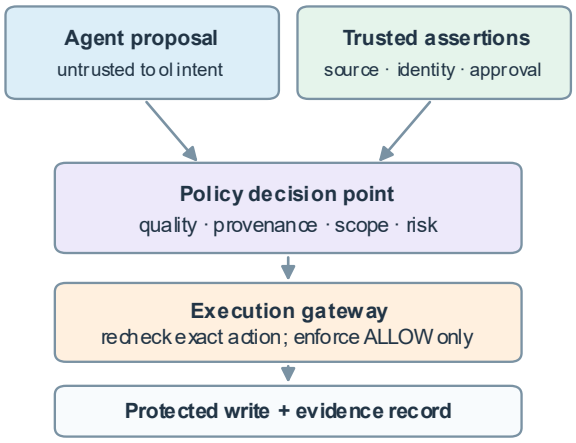

Figure 1. Original architecture: untrusted action proposals, trusted assertions, policy decision and enforced write boundary.

Figure 1 positions an explicit policy and evidence service between agent-produced action proposals and an execution gateway. The ingestion path normalizes a dataset identifier, source attestation, record timestamp, and tenant binding. An identity layer adds principal, delegated role, effective tool set, and resource scope. The agent can select an operation and present evidence identifiers, but it cannot assert its own privileges as authoritative facts.

The gate performs ordered checks: typed schema and maximum evidence age; provenance admissibility; tenant and principal state; resource and tool scope; nonce status; and risk-dependent approval. The execution gateway accepts only an ALLOW decision whose request identity, policy revision, and evidence binding match the operation it is about to perform. In a production implementation, evidence should be bound to a signed, short-lived capability or an authenticated request digest, with atomic nonce consumption and server-side enforcement. These cryptographic and distributed transaction features are architectural requirements, not features tested by the prototype.

A Data Engineering team can implement upstream validation as versioned metadata contracts, but a quality pass is not sufficient to authorize an action. Provenance needs verification against a trusted source registry; freshness needs an operation-specific maximum age; and authorization needs user and workload context. These controls are independent enough to be tested separately, yet should produce a single explainable policy

outcome. The lightweight evaluation encodes these predicates directly in Python and emits local SHA-256 digests that bind each decision to the evaluated input fields.

4.1 Decision representation

Let E denote a proposed action envelope containing the principal, tenant, tool, resource, source evidence, record age, risk category, approval state, nonce and policy revision. The mandatory authorization predicate combines five Boolean checks:

$$H(E) = Q(E) \wedge P(E) \wedge I(E) \wedge S(E) \wedge N(E) \quad (1)$$

Here Q checks required field types, schema validity and freshness; P checks the trusted-source flag; I checks active principal status and the current policy epoch; S checks tenant, tool and resource scope; and N checks the nonce. Each predicate is evaluated for E. The decision function has three outcomes:

$$D(E) \in \{DENY, REVIEW, ALLOW\} \quad (2)$$

Let R(E) identify high-risk operations, A(E) a valid required approval and X(E) a high-risk request with an explicitly expired required approval. The following ordered rules define D(E):

R1 - DENY: if H(E) is false, or X(E) is true.

R2 - REVIEW: if R1 does not apply, R(E) is true and A(E) is false.

R3 - ALLOW: otherwise.

Rules R1–R3 give hard authorization checks precedence over human review. Missing mandatory fields are denied. An absent approval for an otherwise admissible high-risk action triggers REVIEW; an explicitly expired required approval triggers DENY. REVIEW is not permission to execute a tool.

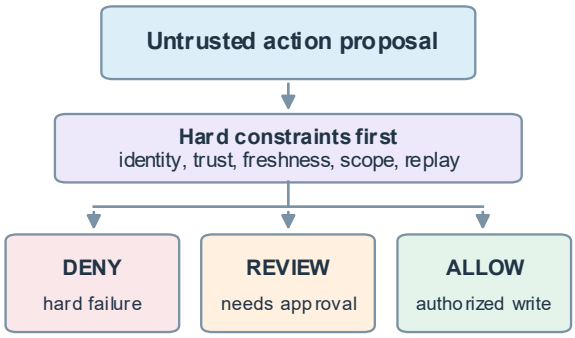

Figure 2. Original pre-execution decision model; REVIEW cannot execute an action.

Figure 2 distinguishes the three outcomes and explicitly separates agent planning from authorized action. A proposed tool call enters the gate regardless of whether it arose from normal reasoning, a misleading document, or an attempted prompt injection. A complete deployment must re evaluate mutable facts immediately before the side effect, to limit time-of-check-to-time-of-use races.

4.2 Decision precedence and an executable policy contract

The ordered rules R1–R3 distinguish an approval that has not yet been granted from one that has explicitly expired. An absent approval triggers REVIEW only after all mandatory checks pass; an expired required approval triggers DENY. This prevents an approval workflow from masking failed source, tenant, identity or tool checks. Equation (1) groups the underlying schema, freshness, source, identity, scope and replay checks; Equation (2) identifies the possible decisions.

A correct deployment must bind the ALLOW decision to the exact principal, canonical request arguments, resource identifier, evidence version, policy revision, expiry, and a single-use operation identifier. The gateway should verify this binding immediately before mutation; an agent must not be able to replay an earlier ALLOW against modified arguments. The Python prototype demonstrates only the deterministic classification and a SHA-256 digest of canonicalized decision fields. It does not mint or verify signed capabilities, consume nonces atomically, or ensure transactional isolation. Those remain enforceable deployment contracts rather than measured implementation features.

4.3 Reference decision procedure and explicit guarantees

The reference procedure consists of seven ordered operations, rather than a learned classifier: (1) reject missing or incorrectly typed required fields; (2) reject evidence older than the configured limit or marked unverified; (3) reject inactive or revoked principals and a mismatched tenant; (4) reject an operation whose tool or resource is outside the principal grant; (5) reject an already consumed nonce; (6) reject an expired approval, or route a high-risk operation without a valid approval to REVIEW; and (7) return ALLOW otherwise, recording a canonical digest of the request and decision. In the executable reference, steps (1)–(6) reside in gate(), whereas gate_with_audit() adds the digest. The functions intentionally do not contact an authorization server or sign records.

This ordering enforces a useful precedence property: a request that violates a hard safety predicate must never be converted into REVIEW merely because an approver might be available. It also exposes a critical interface requirement: the metadata contract must distinguish missing approval from an invalid/expired approval, as those represent different remediation paths. The gate operates over an assumed trusted snapshot. A production write endpoint must ensure the snapshot remains applicable, otherwise authorization at proposal time is only advisory. Returning ALLOW in local code without an enforcing write gateway is not a security outcome.

5. EXPERIMENTAL DESIGN

5.1 Experimental question and controlled scope

The controlled experiment tests whether the proposed deterministic checks reject labeled invalid actions and retain the valid cases, and measures their incremental local processing cost. It does not execute an LLM or invoke real enterprise tools. This distinction matters: a policy engine can classify the cases it was explicitly designed for even when real-world attack discovery, metadata authentication, and integration faults remain unresolved.

Twelve synthetic case families represent two authorized operations, one approval-pending operation, and nine policy violations. Each family contributes 600 instances, giving 7,200 cases. Randomized record ages within 0–299 seconds represent fresh evidence; a stale case has a fixed age of 3,601 seconds. The freshness limit is 900 seconds for illustration, not a recommended general threshold. The dataset is shuffled using fixed seed 20261015 and contains only synthetic identifiers and metadata, with no regulated or customer data.

Table 1. Synthetic scenario families (600 cases per family).

Label	Synthetic case families	Count
ALLOW	Valid low-risk; approved high-risk	1,200
REVIEW	High-risk; approval missing	600

Label	Synthetic case families	Count
DENY	Schema; stale; provenance; cross-tenant; tool; scope; expired approval; revoked principal; replay	5,400

5.2 Baselines and outcome definitions

B0 (no check) forwards every proposed action. B1 (traditional quality check) requires only valid input typing and age not exceeding 900 seconds. B2 (policy-gated) adds source verification, same-tenant binding, active principal, allowlisted tool and resource, unused nonce, approval validity, and an explicit REVIEW route. All three arms receive exactly the same candidate actions. The expected decision for each scenario is defined before the arm is applied.

For each arm, the primary metric is unsafe authorization rate (UAR): the number of ALLOW outcomes on DENY- or REVIEW-labeled proposals divided by all proposals. Exact-decision agreement additionally requires ALLOW, DENY and REVIEW to match their stipulated labels. False rejection means any non-ALLOW on a valid ALLOW-labeled request; review capture is the number of correctly routed REVIEW cases divided by required-review cases. These denominators are deliberately explicit: UAR is a per-proposal synthetic decision error, not an observed frequency of incidents in operation. Latency is reported as p50 and p95 elapsed microseconds from six sequential in-process passes; the full gate includes SHA-256 digest generation while the baselines do not.

5.3 Replication and expected limitations

The complete standard library Python script, scenario-level records, summary table, local evidence hashes, and execution instructions accompany this manuscript as a reproducibility package. The gate and synthetic labels share an explicit policy specification, so agreement measures implementation consistency rather than independently estimated attack resistance. Cases are neither sampled from production incidents nor adaptive adversarial sequences. Single process local timing cannot estimate network round trips, signing, remote policy decisions, database contention, or approval delay.

5.4 Supplemental experiments and analytic safeguards

Two further evaluations examine mechanisms beyond an aggregate score. First, seven single-control ablations independently disable quality, source trust, tenant isolation, tool/resource scope, active-principal checks, replay protection or required approval on the same 7,200 labeled examples. This one-factor-at-a-time design identifies which authored scenario families rely on each predicate; ablation effects are neither independent real-world risk estimates nor additive. Second, a separately generated 2,400-case synthetic stress set contains 600 valid actions, 300 review-required actions and 1,500 hard denials. It probes an exact 900-second freshness boundary, 901-second stale/replay combinations, missing source attestation, tenant/tool compound violations, revoked identity with scope escalation, and hostile text attached to an untrusted source.

The supplemental labels are specified in source code rather than inferred by the gate, but they reflect the same authored policy rules. Thus the suite offers a regression stress test, not an unbiased external ground truth or an adversarial sampling study. For both suites the reported numerator counts unsafe authorizations ALLOW where the stipulated outcome is DENY or REVIEW and the denominator includes all synthetic proposals. Since family frequencies were deliberately balanced, neither percentage estimates the prevalence of harmful real world agent actions. Zero observed violations cannot be used to infer zero risk outside these enumerated conditions.

5.5 Reproducibility, measurement parity and internal validity

Each deterministic run writes case-level predictions, aggregate counts, local evidence digests, and environment metadata. The supplemental evaluator writes counterfactual ablation results and its own case-level decision file; 14 Python unit tests verify selected invariants, edge conditions, and the deliberate inability to recognize prompt injection text independently of source metadata. Independent result validation scripts check the 7,200 primary records and evidence digests as well as the supplemental analysis. On the author's Mac, all 14 unit tests and both result validators passed. None of these checks substitute for an independent security review or code audit. The code uses only the Python standard library and supports execution inside an isolated laptop virtual environment.

The latency comparison intentionally measures unequal work: the policy arm includes canonical serialization and hashing whereas no check and quality only arms omit an equivalent evidence record. Consequently its timing ratio must not be described as the intrinsic cost of individual authorization predicates. Workloads also contain repeated dictionary shapes and cached Python objects, unlike networked execution. The reported author run measurements were collected on macOS 26.5.2 (Apple Silicon ARM64) using Python 3.14.6, across six sequential in-process passes with a monotonic timer; no claim is made about warm/cold caches, concurrency, remote services, or p99 latency. To estimate operational impact, a follow up should execute the same audit path for all arms, collect paired end-to-end timings, and report confidence intervals over independent runs and machines.

6. RESULTS

Table 2. Author reproduced authorization outcomes and local timings (N = 7,200 per arm).

Arm	Unsafe ALLOW	REVIEW	p50 / p95 (µs)
No check	6,000 (83.33%)	0	0.042 / 0.083
Quality only	4,800 (66.67%)	0	0.083 / 0.125
Policy gate	0 (0%)	600	5.167 / 5.750

For class-specific fidelity, the stipulated primary distribution is 1,200 ALLOW (16.67%), 5,400 DENY (75.00%) and 600 REVIEW (8.33%). The unrestricted baseline correctly labels only 1,200 valid cases (16.67% exact agreement). The quality-only arm correctly labels the same 1,200 valid cases and 1,200 schema/staleness denials (33.33% exact agreement); it neither captures any of the 600 required reviews nor resolves the remaining 4,200 hard-denial cases. The full gate matches all 7,200 authored labels (100% exact agreement), including every review-required proposal. These are descriptive agreement values on a deliberately constructed dataset, not predictive classification accuracies for unseen deployments.

Table 2 summarizes the measured reference script outcomes. B0 authorizes all 7,200 requests, including 6,000 that require rejection or review. B1 blocks the 600 malformed and 600 stale records but allows 4,800 requests that violate other rules or require approval. B2 returns ALLOW for the 1,200 valid cases, DENY for 5,400 hard failures, and REVIEW for the 600 high-risk unapproved cases. On this enumerated benchmark, observed unsafe authorization rates are 83.33%, 66.67%, and 0%, respectively. None of the three arms rejects either valid scenario in the synthetic set; only B2 captures all 600 required reviews.

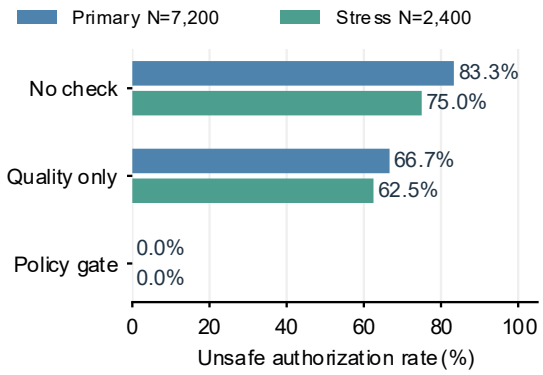

Figure 3. Unsafe authorizations across primary and stress synthetic test suites (percent of all proposals).

On the author's Mac, B2 had a median local invocation time of 5.167 microseconds and a p95 of 5.750 microseconds, versus 0.083 and 0.125 microseconds for B1. B0 measured 0.042 and 0.083 microseconds at p50 and p95, respectively. These small, machine-dependent measurements include timer overhead and unequal instrumentation; B2 also includes canonical serialization and SHA-256 evidence construction. It does not establish acceptable production latency. Every B2 case produces a digest record in the generated evidence file (7,200/7,200, or 100% local evidence generation); the experiment does not test durability, nonrepudiation, signature verification, or distributed audit completeness.

Reducing the policy to typed schema and age screening is consequential in this workload because the other case families remain syntactically correct and fresh. Their failure lies in authority, source trust, tenant binding, replay status, or approval rather than conventional data quality dimensions. The results establish that the extra predicates execute as designed on the synthetic cases; they do not establish a statistically generalizable rate of prevention against agentic attacks.

A strict three-way metric matters because treating REVIEW as merely "not allowed" would conceal whether high-risk requests reach human oversight. Under the reference rules, zero incorrect ALLOW decisions coexists with 600 explicit escalations and 1,200 retained authorized operations. Every primary case receives a local digest (7,200 of 7,200), but a SHA-256 digest is not a signed, durable or independently verifiable audit record.

6.1 Isolating the value of individual controls

Table 3. Counterfactual ablation results (N = 7,200).

Removed control	Unsafe ALLOW	REVIEW
None: full gate	0 (0%)	600
Quality	1,200 (16.67%)	600
Source trust	600 (8.33%)	600
Tenant	600 (8.33%)	600
Tool/resource	1,200 (16.67%)	600
Principal	600 (8.33%)	600
Replay	600 (8.33%)	600
Approval	1,200 (16.67%)	0

Table 3 reports the counterfactual effects of removing one control group at a time from the full policy. Without quality validation or without tool/resource scope, 1,200 of 7,200 synthetic proposals become incorrectly allowed. Removing source trust, tenant isolation, active principal status, or replay checks each admits 600 violating proposals. Suppressing risk-based approval also incorrectly allows 1,200 proposals and eliminates all 600 REVIEW decisions. Each ablation leaves the 1,200 designed legitimate cases allowed, so the extra unsafe decisions are not due to a simple blanket deny policy. These figures reflect exactly which synthetic families were written to exercise each predicate; they should not be interpreted as real-world marginal risk or combined additively.

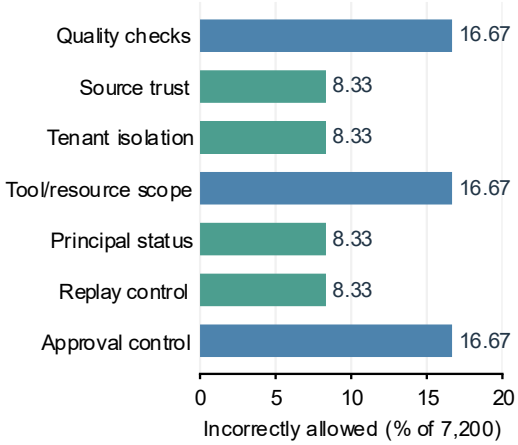

Figure 4. Counterfactual unsafe authorization rate when one policy control is removed (synthetic primary suite).

6.2 Boundary and compound failure stress results

Table 4. Supplemental stress results (N = 2,400).

Arm	Unsafe ALLOW	Valid ALLOW	REVIEW
No check	1,800 (75%)	600	0
Quality only	1,500 (62.5%)	600	0
Policy gate	0 (0%)	600	300

Table 4 presents the second, separately generated synthetic set of 2,400 requests. The unchecked baseline incorrectly allows 1,800 invalid or approval required proposals (75.0%); schema plus age incorrectly allows 1,500 (62.5%). The full gate returns 600 ALLOW, 1,500 DENY, and 300 REVIEW as prespecified. All 300 records exactly at the 900 second freshness limit are accepted; records at 901 seconds with a replay violation are rejected. Missing source metadata fails closed, and simultaneous tenant/tool or identity/scope violations do not route to human approval before basic authorization passes.

A special stress family contains injection-like text and an independently marked unverified source. Its rejection demonstrates a source attestation predicate, not semantic identification of malicious language: if the source attestation flag is artificially changed to verified while other metadata remain admissible, the reference gate allows the proposal. This controlled counterexample is retained in the automated tests to prevent a misleading interpretation of the result. The studies together show implementation consistency over two controlled synthetic distributions; neither uses a live LLM, an adaptive attacker, or trusted attestations backed by cryptography.

Across the stress distribution, class-specific accounting again shows preserved utility: all 600 valid actions are allowed by every arm, but only the full gate sends the 300 approval-dependent proposals to REVIEW and denies all 1,500 stipulated hard failures. Thus the full gate has 100% exact-label agreement on this authored set, compared with 25.0% for no checks (600/2,400) and 37.5% for quality-only screening (900/2,400).

These values summarize the test fixtures, including compound cases, rather than independently sampled attacker success probabilities.

6.3 Evaluation synthesis and interpretive boundaries

The evaluations cover five complementary dimensions: (i) exact ALLOW/DENY/REVIEW fidelity, (ii) false rejection of legitimate actions, (iii) escalation capture, (iv) single-predicate sensitivity and compound boundary cases, and (v) local processing time and evidence-record generation. Their strongest finding is internal consistency under precisely enumerated rules. The absence of incorrect authorizations reflects both the gate design and the scenario construction, which share a specification. There is no independent human-labeled action corpus, model-generated tool trace, live executor, concurrent state change, or adaptive attacker. Statistical confidence bounds would therefore misrepresent the designed and correlated test population; meaningful generalization requires new, independently sampled cases and independent adjudication.

7. DISCUSSION AND PRACTICAL IMPLICATIONS

Three operational lessons follow. First, an enterprise agent should submit an action intent and verifiable evidence references, not a self-issued claim that its action is safe. Second, the policy decision point must be independent of the model and the policy enforcement point must protect the actual mutation API. Third, operators need to distinguish failed data validation from missing user authority and from an approval dependent action. The DENY/REVIEW/ALLOW distinction improves audit semantics and avoids reducing governance to a single confidence score.

For regulated organizations, deployment should begin with low-risk case updates whose identities, resource scopes, and allowed APIs are already managed outside the agent. A source registry can capture the approved producer and dataset version; a pipeline contract can bind record timestamps and schema versions; an access service can resolve current workload identity and tenant; and an approval service can record a purpose bound attestation. The gateway should re-check mutable authorization state before writing. Temporary tool credentials, network segmentation, least privilege, and human escalation remain necessary defense in depth measures.

The local evidence hash is an integrity convenience, not a standalone audit system. A viable production design would retain canonicalized request and decision inputs in append-only storage; bind the decision to source version and policy version identifiers; sign or authenticate records at a trusted boundary; and log execution results under the same correlation identifier. Operational observability should report deny reasons, approval latency, stale source frequency, cross tenant attempts, policy service failures, and unprotected tool bypass attempts without disclosing sensitive record contents.

A separate business tradeoff is availability. Fail-closed enforcement can block legitimate actions during metadata outages. A safe operating model requires explicit exception governance, queueing for approval, clear timeouts, tested recovery procedures, and an incident process for policy revisions. A broad exception that bypasses provenance or authorization would undermine the control. Policy coverage should be exercised against alternative tools, indirection through secondary APIs, simultaneous requests, partial execution, and changes to user authority during long-running workflows.

7.1 Practical enterprise execution example

Consider a case management agent asked to update an exception review status. The Data Engineering pipeline supplies a record version, ingestion lineage, and timestamp. The orchestration service supplies a proposed update to case C-42, the delegated principal, and an action correlation ID. An independently operated authorization service resolves the principal's tenant and grants; the policy service compares them with C 42 and the allowed tool set. A correctly scoped, fresh low-risk update may receive ALLOW. If the source record was superseded, the action is DENY even if the natural language reasoning is persuasive. If the requested transition is restricted, a missing approved ticket leads to REVIEW with no mutation. None of these decisions depends on the model's stated certainty.

The actual gateway must use short-lived identity credentials and a narrowly scoped executor identity, with network controls preventing an ungated alternative API route. The sequence should be: construct canonical request; query trusted metadata and identity services; compute policy decision; bind a short-lived authorization object to the exact request; at the writer, recheck policy revision, nonce, and current authorization in the same transactional boundary where possible; then record either the side effect or a durable failure outcome. If the backend cannot atomically consume the nonce with the write, idempotency keys and a transactional outbox can reduce not eliminate replay and partial failure risks.

7.2 Governance and operational observability

Ownership should be explicit. Data owners establish authoritative source registries and freshness service objectives; platform teams operate the gate and protect enforcement routes; security owners publish version-controlled tool and resource policies; application owners classify high-risk transitions; approvers authorize exceptions with purpose, expiry, and named scope. Continuous controls include negative integration tests against each mutation API, sampled comparison between gate decisions and backend audit logs, emergency policy revocation exercises, and alerts on changes in DENY, REVIEW, and timeout rates. Approval backlogs must be measured separately from automated decision latency because a fast policy evaluator says little about end-to-end human workflow time.

7.3 Traceability from threat to authoritative control

Table 5. Authoritative owner and enforcement boundary.

Failure	Authority	Enforcement
Stale/unknown data	Source registry	Policy gate
Tenant/scope	Identity service	Write API
Unauthorized tool	API grants	Write API
High-risk action	Approval service	Write API
Replay/expiry	Nonce/epoch store	Transactional API
Bypass route	Network policy	Backend API

Table 5 connects each failure mode to an independently managed control. This matters because assigning every check to the Data Engineering service would concentrate authority and create a bypass opportunity. Data Engineering can own the existence, timeliness, and producer identity of evidence; security services must attest to principal authority; human reviewers must attest to approvals; and application gateways must enforce the final decision on a side-effecting endpoint. Correlation IDs should join these separate records without exposing sensitive contents in general purpose logs. The service contract should also specify error semantics: an unavailable attestation, stale authorization epoch, or unresponsive decision service is not equivalent to a

successful policy outcome.

An actionable rollout sequence is: begin with a single mutation API in audit-only mode; compare predicted DENY/REVIEW against owner-approved ground truth; introduce shadow policy revisions and test their false rejection rates; protect the write endpoint with mandatory enforcement and explicit service identities; then test direct API bypass, concurrent replay, policy changes between check and write, and loss of metadata services. This progression can identify incorrect policy design while minimizing the risk that automated restrictions interrupt legitimate work. It is an implementation recommendation, not a process evaluated by the supplied prototype.

8. LIMITATIONS AND FUTURE WORK

The original and supplemental suites test constructed labels derived from an authored policy contract; even the separately generated stress cases are not an independently adjudicated challenge set. The design assumes trustworthy metadata and static tool identities; it has no live LLM, adaptive attacker, forged signatures, concurrent approval races, remote policy service, or real API mutations. Thus 0% synthetic UAR and exact-label agreement do not imply zero risk in an enterprise deployment. Local timing is asymmetric because only the full gate creates digest evidence, and the 5.750-microsecond p95 excludes all remote operations and approval delay. Further limitations include fixed scenario frequencies, repeated in-process objects, and no stratified error analysis over real users or applications. Accordingly, the next evaluation should include externally authored adversarial cases, blinded expert labels, protected backend enforcement tests and multi-host timing with matched audit work across arms.

Future work should connect the gateway to actual tool-using agents and a policy engine, evaluate attempted bypasses under AgentDojo-style tasks, and include a trusted provenance verifier and atomic action token service. Further experiments should report attack success rate with adaptive attackers, per class false positives and negatives, request throughput, remote p95/p99 latency, cost, approval burden, audit verifiability, and performance during source and policy service failures. Domain experts should assess whether the proposed scopes encode genuine business permission rather than merely matching syntactic labels.

9. CONCLUSION

AI agents change the consequences of data misuse because retrieved evidence can trigger mutations rather than merely influence generated text. This study makes the boundary actionable by specifying a policy-gated data-to-action contract with independently trusted assertions, deterministic DENY/REVIEW/ALLOW precedence, and mandatory enforcement at the application write API. In the 7,200-case primary suite, the gate exactly reproduces 1,200 allowed, 5,400 denied and 600 review-required outcomes; a quality-only comparator still authorizes 4,800 invalid or approval-dependent proposals. A further 2,400 synthetic cases confirm the specified boundary and compound-failure behavior, while seven ablations show which controls account for different authored failure classes. Two visual comparisons and readable tables expose this result more clearly than a single aggregate rate. The findings substantiate a reproducible policy-engineering reference implementation, not demonstrated resistance to live prompt injection or production attackers. Future research should independently label tool-intent traces from operationally realistic agent tasks, vary source and identity attestation failures, and evaluate policy decisions under concurrent requests, stale epochs, forged evidence, and bypass attempts. A production prototype must add signed action binding, atomic nonce consumption, a protected execution gateway, and durable audit correlation. Experiments should measure false rejections, review workload, attack success, throughput, and end-to-end p95/p99 latency under network and service failures. These steps will test whether policy gating improves secure agent autonomy without unacceptable availability or human-approval costs.

10. REFERENCES

[1] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In Proceedings of the 11th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=WE_vluYUL-X.

[2] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. Advances in Neural Information Processing Systems, 36, 68539–68551. https://doi.org/10.52202/075280-2997.

[3] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In Proceedings of ACM CCS 2023. https://doi.org/10.1145/3605764.3623985.

[4] Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., and Tramèr, F. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. Advances in Neural Information Processing Systems, 37, 82895–82920. https://doi.org/10.52202/079017-2636.

[5] OWASP GenAI Security Project. 2025. OWASP Top 10 for Agentic Applications for 2026, December 9. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/.

[6] Gautam, A. R. 2026. Data Engineering for Clean Context Pipelines: Advancing Reliability, Efficiency, and Cost Effectiveness in LLM-Assisted Software Development. International Journal of Computer Applications, 187(105), 21–26. https://doi.org/10.5120/ijcacc5cac127607.

[7] Gautam, A. R. 2025. Impact of High Data Quality on LLM Hallucinations. International Journal of Computer Applications, 187(4), 35–39. https://doi.org/10.5120/ijca2025924909.

[8] Cravero, A., Díaz-Villegas, J., and Xiao, Z. 2026. From Data Quality to Quality of Agentic Data Use: A Conceptual Framework for Agentic Data Engineering. Applied Sciences, 16(17), 8887. https://doi.org/10.3390/app16178887.

[9] Autio, C., Schwartz, R., Dunietz, J., Jain, S., Stanley, M., Tabassi, E., Hall, P., and Roberts, K. 2024. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.600-1.

[10] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. 2020. Zero Trust Architecture. NIST Special Publication 800-207. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207.

[11] Lebo, T., Sahoo, S., and McGuinness, D., eds. 2013. PROV-

O: The PROV Ontology. W3C Recommendation, 30 April 2013. https://www.w3.org/TR/prov-o/.

[12] Open Policy Agent Project. 2026. Open Policy Agent Documentation: Policy Decision and Enforcement Architecture. Cloud Native Computing Foundation project documentation. https://www.openpolicyagent.org/docs/.

[13] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. 2024. Lost in the Middle: How Language Models Use Long Contexts. Transactions of the Association for Computational Linguistics, 12, 157–173. https://doi.org/10.1162/tacl_a_00638.