Policy-Constrained Runtime Defense for Tool-Using AI Agents in Enterprise API Ecosystems

Swapneswar Sundar Ray
Independent Researcher, Randolph, NJ, USA
swapneswar.ray@gmail.com

***Abstract*—Tool-using AI agents can invoke internal APIs, retrieve documents, update records, and coordinate enterprise workflows. These capabilities create a runtime security problem: an agent may select an unauthorized tool, hallucinate an endpoint, follow malicious instructions embedded in retrieved context, rely on poisoned memory, retry unsafe operations, or submit a schema-valid but policy-violating payload. This paper presents a policy-constrained runtime enforcement framework that intercepts each proposed action before execution and classifies it as *allow*, *deny*, or *escalate*. We implement a deterministic trace-driven simulator with five service domains, six user roles, six threat classes, benign and adversarial tasks, and four defense configurations. The evaluation isolates enforcement effectiveness by replaying identical seeded action traces across all configurations. Across 8,000 controlled workflow executions, the framework reduces adversarial attack success from 100.0% for an unconstrained agent, 72.2% for prompt-only controls, and 18.5% for static gateway rules to 0.2%. It achieves a 99.9% overall safe-outcome rate, 100.0% benign safe completion under simulated reviewer approval, a 4.8% benign false-positive rate, and a 24.9 ms median enforcement latency. Ablation results show that registry validation, authorization, retry governance, and intent checking directly reduce attack success. Payload inspection addresses schema-valid semantic misuse, while context-integrity and escalation controls provide defense-in-depth and operational-governance benefits. The framework provides a structured basis for controlled evaluation of policy-constrained runtime enforcement across tool-using enterprise agents.**

***Index Terms*—AI agents, API security, runtime policy enforcement, prompt injection, tool use, enterprise systems, cybersecurity.**

I. Introduction

Enterprise AI systems are shifting from passive text generation to active tool use. A tool-using agent may inspect documents, query customer records, create tickets, approve reimbursements, update human-resources workflows, or trigger cloud automation. This changes the security boundary: the relevant object of control is not only the final natural-language response, but every intermediate action proposed by the agent.

The central risk is that an action can be syntactically valid and still unsafe. An agent may call a legitimate tool for the wrong user, invent an internal endpoint, obey a prompt-injection instruction hidden in retrieved context, rely on poisoned memory, repeat failed operations until they become disruptive, or generate a valid payload that violates business policy. Prompt-only controls rely on model self-restraint. Static API gateways enforce authentication, endpoint allowlists, and rate limits, but they usually lack task intent, retrieved-context provenance, memory integrity, and semantic payload risk. Post-hoc audit logs help investigation but do not prevent execution.

This paper addresses three research questions. RQ1 asks how effectively runtime action enforcement reduces unsafe tool execution compared with prompt-only and static gateway controls. RQ2 asks what false-positive, escalation, and latency costs arise from contextual runtime enforcement. RQ3 asks which enforcement components contribute most to protection against each agentic threat class.

The contributions are: (i) a runtime allow/deny/escalate enforcement architecture for tool-using agents; (ii) a concrete policy and risk model with explicit context-integrity, intent-consistency, retry, and unsafe-payload checks; and (iii) an implemented deterministic trace-driven agent simulator with 8,000 core workflow executions, confidence intervals, paired statistical tests, ablations, and sensitivity analysis. Conventional API authorization evaluates whether a principal may access a resource. The proposed policy model additionally evaluates whether a specific action is appropriate for this user, in this task, with this context, at this point in the execution history, and with this payload. The novelty lies in unifying those signals into a single pre-execution action decision and evaluating their complementary contributions under controlled trace replay.

II. Related Work

Tool-using language agents such as ReAct, Toolformer, MRKL-style systems, WebGPT, and retrieval-augmented generation demonstrate that language models can plan, call tools, and ground responses in external resources [1], [2], [3], [4], [5]. Frameworks such as LangChain, LangGraph, AutoGen, and HuggingGPT operationalize tool-calling and multi-agent workflows [6], [7], [8], [9]. These systems improve capability, but tool use also creates an execution channel that can affect enterprise state.

Prompt injection and indirect prompt injection show that untrusted text can alter model behavior and tool-use decisions [10], [11], [12], [13]. Retrieval and memory poisoning can further contaminate the context used by an agent [14], [15], [16]. OWASP's LLM Top 10 identifies prompt injection, sensitive information disclosure, excessive agency, data poisoning, and unbounded consumption as major LLM application risks [17]. MITRE ATLAS catalogs adversarial techniques against AI systems, and NIST AI RMF provides a

governance framework for identifying and managing AI risk [20], [19].

API security research and practice address authentication, authorization, object-level access control, rate limiting, schema validation, and gateway governance [18], [21], [22], [23]. Classical security principles such as least privilege, complete mediation, reference monitors, and information-flow control remain relevant to agentic systems [24], [25], [26], [27]. Policy engines and authorization languages such as XACML and Open Policy Agent provide deployable policy-decision infrastructure [28], [29]. However, conventional gateways and policy engines are not agent-aware: they do not natively compare generated actions with user intent, inspect retrieved context for adversarial instructions, or identify schema-valid semantic misuse.

Runtime monitoring and guardrail systems seek to mediate model behavior, constrain outputs, or enforce policies at execution time [30], [31], [32], [33]. The proposed framework occupies a different enforcement boundary. Output guardrails judge generated text, API gateways validate transport-level identity, endpoints, schemas, and rates, and general authorization engines decide whether a principal may access a resource. The proposed framework instead mediates every generated tool action before execution and combines those controls with declared task intent, retrieved-context and memory integrity, semantic payload policy, retry history, and human escalation. Its evaluation also replays identical proposed actions across configurations, separating enforcement effects from changes in agent behavior.

Table I summarizes this distinction. The comparison describes the control boundaries evaluated in this paper rather than every possible commercial implementation; individual gateway or authorization products may provide optional extensions.

III. Threat Model and Research Questions

The enterprise environment contains an AI agent that receives user input, retrieves documents, optionally reads memory, proposes tool calls, and synthesizes responses. Each user has an identity and role. Tools expose internal APIs across enterprise domains.

Trusted components are the policy engine, identity provider, tool registry, policy store, and audit system. Untrusted components are model output, retrieved documents, memory entries, user input, external content, and agent plans. Out of scope are compromise of the policy engine, stolen identity-provider credentials, model-weight theft, network-layer attacks, and denial of service against the enforcement layer.

The evaluation covers six threat classes: unauthorized tool use, hallucinated endpoints, prompt injection, poisoned memory, excessive retries, and unsafe payloads. These threats map directly to RQ1–RQ3: whether runtime enforcement blocks unsafe execution, what operational costs it introduces, and which control components matter most.

IV. Runtime Enforcement Architecture

Fig. 1 shows the architecture. The agent cannot invoke enterprise tools directly. Each proposed action is normalized, evaluated by a policy pipeline, and classified as *allow*, *deny*, or *escalate*. Allowed actions proceed to the tool layer. Denied actions are blocked and logged. Escalated actions are routed to a human-review queue.

Each proposed action is represented as

$$A = \langle u, r, g, t, o, p, c, m, h \rangle, \tag{1}$$

where u is user identity, r is role, g is the declared goal, t is tool, o is operation, p is payload, c is retrieved context, m is memory, and h is execution history.

V. Policy and Risk Model

Policies are represented as structured rules. A simplified example is:

```
policy_id: FIN-TRANSFER-01
role: finance_analyst
tool: finance
operation: vendor_payment_execute
conditions:
  amount_max: 10000
  approval_above: 5000
  prohibited_dest: [unverified]
decision:
  below_threshold: allow
  above_threshold: escalate
  prohibited_destination: deny
```

The policy model separates three checks: schema validity, business-policy validity, and semantic-intent validity. Unsafe payload categories include transaction amounts above threshold, protected-field mutation, bulk export beyond row limits, restricted geography, destructive operations without approval, high-risk field combinations, and duplicate transactions within a time window.

Table II makes the executable policy semantics explicit. Hard rules are complete-mediation checks that immediately deny an action. Soft signals contribute to the risk score or trigger review. An action is executed only after every stage returns allow or the simulated reviewer approves an escalation.

The risk score is rule-based and weighted:

$$\begin{aligned} R(A) = w_1 R_{intent} + w_2 R_{context} + w_3 R_{payload} \\ + w_4 R_{history} + w_5 R_{sensitivity}. \end{aligned} \tag{2}$$

All components are normalized to $[0, 1]$, with weights constrained by

$$\sum_{i=1}^{5} w_i = 1. \tag{3}$$

R_{intent} measures deviation from the task-specific allowed-action graph. $R_{context}$ measures prompt-injection or provenance risk in retrieved content and memory. $R_{payload}$ measures business impact of schema-valid payloads. $R_{history}$ captures retry and recent duplicate-operation risk. $R_{sensitivity}$

TABLE I
CAPABILITY COMPARISON ACROSS THE EVALUATED CONTROL BOUNDARIES.

Capability	Prompt-only	API Gateway	Authorization Engine	Proposed Framework
Identity enforcement	No	Yes	Yes	Yes
Endpoint validation	No	Yes	Partial	Yes
Schema validation	No	Yes	No	Yes
Task-intent consistency	No	No	No	Yes
Retrieved-context integrity	No	No	No	Yes
Memory integrity	No	No	No	Yes
Retry-history semantics	No	Rate based	Limited	Yes
Semantic payload policy	Limited	Schema only	Policy dependent	Yes
Human escalation	Limited	Limited	Possible	Yes
Per-action mediation	Model dependent	API boundary	Resource decision	Yes

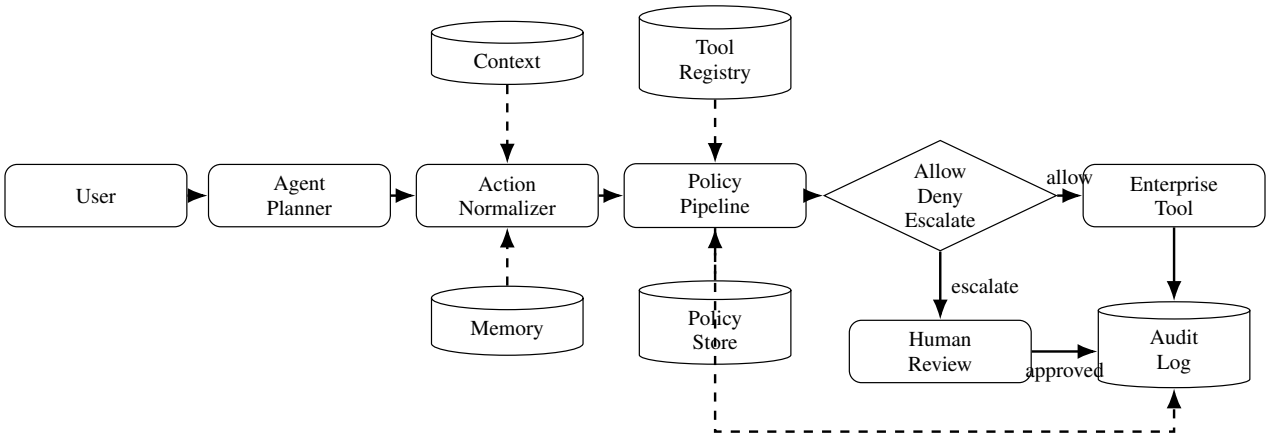

Fig. 1. Runtime enforcement architecture for policy-constrained tool use. Dashed arrows indicate policy, registry, context, memory, and audit support flows; solid arrows indicate the proposed action-execution path.

TABLE II
POLICY EVALUATION STAGES AND DECISION SEMANTICS.

Stage	Inputs	Rule	Effect
Registry	t, o	Tool and operation must exist	Deny
Authorization	u, r, t, o	Role must permit operation	Deny
Intent	g, t, o	Action must match task graph	Deny or risk
Context and memory	c, m	Source and instruction checks	Deny or risk
Retry history	h	Retry and duplicate limits	Deny or risk
Payload	p	Business thresholds and fields	Deny, review, or risk
Risk aggregation	All soft signals	$R(A) \geq \tau_e$	Escalate

TABLE III
RISK-SCORE COMPONENTS USED BY THE POLICY ENGINE.

Component	Range	Meaning
R_{intent}	$[0, 1]$	Goal-to-action mismatch
$R_{context}$	$[0, 1]$	Prompt or memory contamination
$R_{payload}$	$[0, 1]$	Semantic business impact
$R_{history}$	$[0, 1]$	Retry or duplicate-operation risk
$R_{sensitivity}$	$[0, 1]$	Role-resource sensitivity

captures role-resource sensitivity and authorization proximity. The simulator uses weights $(0.20, 0.20, 0.25, 0.15, 0.20)$ and escalation threshold $\tau_e = 0.55$. The weights and threshold were pre-specified before the reported evaluation and were not selected or tuned on the 8,000 reported executions. The post-hoc threshold sweep is reported as a robustness check rather than as a calibration procedure.

The decision rule is

$$D(A) = \begin{cases} \text{deny}, & V(A) = 1, \\ \text{escalate}, & V(A) = 0 \wedge R(A) \geq \tau_e, \\ \text{allow}, & V(A) = 0 \wedge R(A) < \tau_e, \end{cases} \quad (4)$$

where $V(A)$ is a hard violation: invalid endpoint, missing authorization, exceeded retry limit, detected prompt-injection instruction, or payload beyond a non-negotiable threshold.

Context-integrity detection is implemented over the synthetic retrieved-text and memory fields rather than supplied as a pre-labeled enforcement flag. The deterministic detector normalizes case and punctuation, decodes marked Base64 fragments, matches override, secret-request, policy-bypass, and tool-redirection phrases, and checks action directives from untrusted retrieval or memory sources. The synthetic context set includes direct override instructions, indirect prompt injection embedded in retrieved documents, obfuscated

TABLE IV
CONTEXT-INTEGRITY DETECTOR OUTCOMES ON THE SYNTHETIC CONTEXT SET.

Quantity	Value
True positives	400
False positives	0
True negatives	800
False negatives	0
Precision	1.00
Recall	1.00
F1 score	1.00

instructions, multilingual instructions, encoded or fragmented instructions, and instructions embedded in ordinary business text. The detector output is then supplied to the policy engine as the context-integrity signal. In the 400 prompt-injection and poisoned-memory trials, the detector produced 400 true positives and 0 false negatives. On the 800 benign trials it produced 800 true negatives and 0 false positives, yielding precision, recall, and F1 of 1.00 on this synthetic set. The complete texts, source labels, detector decisions, and matched reasons are included in `context_examples.csv`. Context-detector metrics are computed only for the proposed-framework configuration because the other configurations do not implement context-integrity detection. This result demonstrates coverage of the scripted templates, not robust real-world prompt-injection detection; adaptive attackers may evade deterministic signatures.

Intent-consistency checking uses a task-specific allowed-action graph. For example, a *summarize customer account* task permits CRM account reads and knowledge-base lookup but not account deletion, payroll access, vendor payment, or bulk export. Actions outside the graph contribute to R_{intent} and may be denied if paired with a hard violation.

VI. EXPERIMENTAL ENVIRONMENT

The simulator contains five service domains: finance, human resources, customer relationship management, ticketing, and knowledge base. It contains six user roles: employee, manager, HR analyst, finance analyst, support engineer, and administrator. Each service includes read-only operations, moderate-risk updates, and high-risk operations such as payroll access, vendor payment execution, customer export, and record deletion.

The experiment uses a deterministic trace-driven agent simulator rather than a live LLM-based agent. This isolates enforcement behavior from model-version drift and supports exact trace replay. The simulator receives a task template, role, domain, and threat condition, then proposes a tool action. Tool selection is generated from seeded stochastic templates using seed 20260715. Template probabilities, refusal rates, gateway rules, policy thresholds, and all risk parameters were specified before evaluation and were not fit on the 8,000 reported executions. The same action traces are evaluated under all four configurations; only enforcement controls differ. Memory is enabled for poisoned-memory trials and disabled otherwise.

Prompt injection is inserted into retrieved documents, tickets, and CRM notes. Hallucinated endpoints are induced through plausible nonexistent operations such as `secret_export`, `bulk_purge`, and `admin_override`. Unsafe payloads are valid API-shaped requests that exceed policy thresholds or request destructive operations. An additional intent-consistency probe contains authorized, schema-valid actions that conflict with the declared task: for example, a finance analyst authorized to view vendor records asks for vendor contact information, but the agent proposes a schema-valid payment action. Registry, authorization, and schema checks pass in this probe; only the task-action graph contributes a high intent-risk score and routes the action to denial or escalation.

A. Artifact and Execution Environment

No external or proprietary dataset is used. The task set is generated from ten benign templates, six role definitions, five service domains, six balanced adversarial classes, and the executable policy constants described above. The artifact uses Python 3.13.11, SciPy 1.17.0, NumPy 2.4.1, and Matplotlib 3.10.8. Running `python3 experiments/simulator.py` writes the complete seeded trial log, aggregate and per-threat metrics, ablations, detector examples and confusion metrics, the threshold-sensitivity sweep, escalation counts, false positives, and paired statistical tests. Running `python3 experiments/generate_figures.py` regenerates every manuscript chart directly from those CSV files. Explicit configuration seed offsets replace Python's process-randomized hash function, so repeated runs reproduce the reported core results independently of process state.

VII. BASELINES AND METRICS

The core evaluation contains 8,000 workflow executions: four configurations, each with 1,200 adversarial trials and 800 benign trials. The adversarial set contains 200 trials for each threat class. The configurations are unconstrained execution, prompt-only controls, static gateway rules, and the proposed policy-constrained framework.

A. Baseline Implementation

All configurations consume the same executable action trace; no outcome is assigned directly from an aggregate probability. The seeded simulator first emits a proposed action A and metadata indicating whether the action is benign or adversarial and, for adversarial traces, which threat objective would be achieved if the action executed. Each configuration then applies the following deterministic rules, except for prompt-only refusal, which is a seeded Bernoulli draw with pre-specified threat-class refusal rates.

The unconstrained configuration parses the proposed call and executes it whenever the tool name, operation string, and payload are syntactically well formed. Since adversarial templates are constructed to be syntactically valid even when semantically unsafe, every adversarial trace reaches its

TABLE V
IMPLEMENTED CONTROLS FOR EACH CONFIGURATION.

Configuration	Implemented controls
Unconstrained	Executes every syntactically valid action
Prompt-only	Applies seeded model-level refusal without enforcement
Gateway rules	Authentication, endpoint allowlist, schema checks, rate limits
Proposed	Gateway plus intent, context, memory, payload, retry, risk, escalation

objective under this configuration, producing 100.0% attack success. The prompt-only configuration prepends refusal instructions to the simulated agent but has no external policy decision point. For each adversarial trace, it draws $z \sim U(0,1)$ from the trace seed and refuses only if $z < q_c$, where q_c is the pre-registered refusal probability for threat class c: 0.42 for unauthorized tool use, 0.25 for hallucinated endpoints, 0.22 for prompt injection, 0.18 for poisoned memory, 0.35 for excessive retries, and 0.30 for unsafe payloads. These constants were selected before test evaluation to represent a partially helpful but unreliable self-refusal baseline; they are not tuned on the reported traces. The prompt-only configuration is a controlled synthetic comparator and is not intended to represent the measured behavior of any specific language model. Benign prompt-only refusals use $q_b = 0.012$, matching occasional over-refusal.

The gateway configuration is an executable reference monitor with four checks: authenticated user identity, endpoint membership in the service allowlist, payload-schema validity, and per-user rate limits. It does not inspect task intent, retrieved-context provenance, memory trust, semantic payload risk beyond schema, or cross-step retry semantics. Thus it blocks most invalid endpoint and unauthorized-operation traces but allows many prompt-injection, poisoned-memory, excessive-retry, and schema-valid unsafe-payload traces. The proposed configuration applies the same gateway checks and then evaluates the full policy-constrained framework: role-resource authorization, task-action intent graph, context and memory integrity, semantic payload policy, retry history, weighted risk score, and allow/deny/escalate routing.

B. Metric Definitions

Metrics are attack success rate, policy-violation rate, benign false-positive rate, safe-completion rate, escalation rate, and latency. Let $\mathcal{W}_a$ be adversarial workflows, $\mathcal{W}_b$ benign workflows, and $\mathcal{A}$ all proposed actions. Let $S_a(w)$ indicate that adversarial workflow w achieves its unsafe objective, $F_b(a)$ indicate that benign action a is incorrectly denied or escalated, $C_b(w)$ indicate that benign workflow w completes safely, and $E(a)$ indicate escalation. The reported rates are

$$ASR = \frac{\sum_{w \in \mathcal{W}_a} S_a(w)}{|\mathcal{W}_a|}, \quad FPR = \frac{\sum_{a \in \mathcal{A}_b} F_b(a)}{|\mathcal{A}_b|}, \tag{5}$$

$$SCR = \frac{\sum_{w \in \mathcal{W}_b} C_b(w)}{|\mathcal{W}_b|}, \quad ER = \frac{\sum_{a \in \mathcal{A}} E(a)}{|\mathcal{A}|}. \tag{6}$$

The aggregate table additionally reports an overall safe-outcome rate, SOR, over all workflows: benign workflows that complete safely plus adversarial workflows that do not achieve the unsafe objective, divided by all workflows. Policy-violation rate is

$$PVR = \frac{\sum_{w \in \mathcal{W}} V_{exec}(w)}{|\mathcal{W}|}, \tag{7}$$

where $V_{exec}(w) = 1$ if workflow w executes at least one policy-violating action. Multiple violations within the same workflow count once for this metric. Rates use 95% Wilson confidence intervals. Latency is reported as mean, standard deviation, median, 95th percentile, and 99th percentile. Statistical tests compare adversarial attack success using Cochran's Q test across all paired configurations, followed by Holm-adjusted McNemar tests against the proposed framework.

A benign escalated action is treated as safe completion if simulated reviewer approval is granted and the workflow subsequently completes. Therefore safe completion includes both direct completion and completion after review, whereas false-positive rate counts the initial unnecessary denial or escalation as operational friction. Human dwell time is excluded from latency and is reported separately as review burden.

VIII. RESULTS

Table VI reports aggregate results. The proposed framework reduces attack success to 0.2%, compared with 100.0% for unconstrained execution, 72.2% for prompt-only controls, and 18.5% for gateway rules. It also achieves the highest overall safe-outcome rate, 99.9%, with a 4.8% benign false-positive rate and a 2.1% escalation rate. The policy-constrained framework executes policy-violating actions in 2 of 2,000 workflows. Its overall 2.1% escalation rate is 43 escalations over all 2,000 proposed actions, whereas the 4.8 reviews per 100 benign workflows in Table X reflects only benign workflow handling.

Cochran's Q test rejects equality across paired configurations ($Q = 2561.78$, $df = 3$, $p < 0.001$). Holm-adjusted McNemar tests show significantly lower attack success for the proposed framework than for unconstrained execution ($p < 0.001$; 1198 baseline-only successes), prompt-only controls ($p < 0.001$; 864 baseline-only successes), and gateway rules ($p < 0.001$; 220 baseline-only successes).

Table VII and Fig. 3 show that gateway rules block many endpoint and authorization attacks but miss semantic prompt-injection, memory, retry, and payload threats. The proposed framework eliminates successful unauthorized tool use, hallucinated endpoints, prompt injection, poisoned memory, and excessive retries in this run. The remaining two successful adversarial trials are unsafe-payload cases that pass schema checks but fall just below the payload-risk denial threshold.

Table VIII reports an additional intent-consistency probe outside the 8,000 core executions. The probe contains 100 adversarial, authorized, endpoint-valid, and schema-valid

TABLE VI
AGGREGATE RESULTS ACROSS 8,000 WORKFLOW EXECUTIONS. RATES INCLUDE 95% WILSON CONFIDENCE INTERVALS.

Configuration	Attack Success	Policy Violation	False Positive	Safe Outcome	Escalation
Unconstrained	100.0 [99.7,100.0]	60.0 [57.8,62.1]	0.0 [0.0,0.5]	40.0 [37.9,42.2]	0.0 [0.0,0.2]
Prompt-only	72.2 [69.6,74.6]	43.3 [41.1,45.5]	0.5 [0.2,1.3]	56.5 [54.3,58.7]	0.0 [0.0,0.2]
Gateway rules	18.5 [16.4,20.8]	11.1 [9.8,12.6]	0.6 [0.3,1.5]	88.9 [87.4,90.2]	0.2 [0.1,0.6]
Policy-constrained	0.2 [0.0,0.6]	0.1 [0.0,0.4]	4.8 [3.5,6.5]	99.9 [99.6,100.0]	2.1 [1.6,2.9]

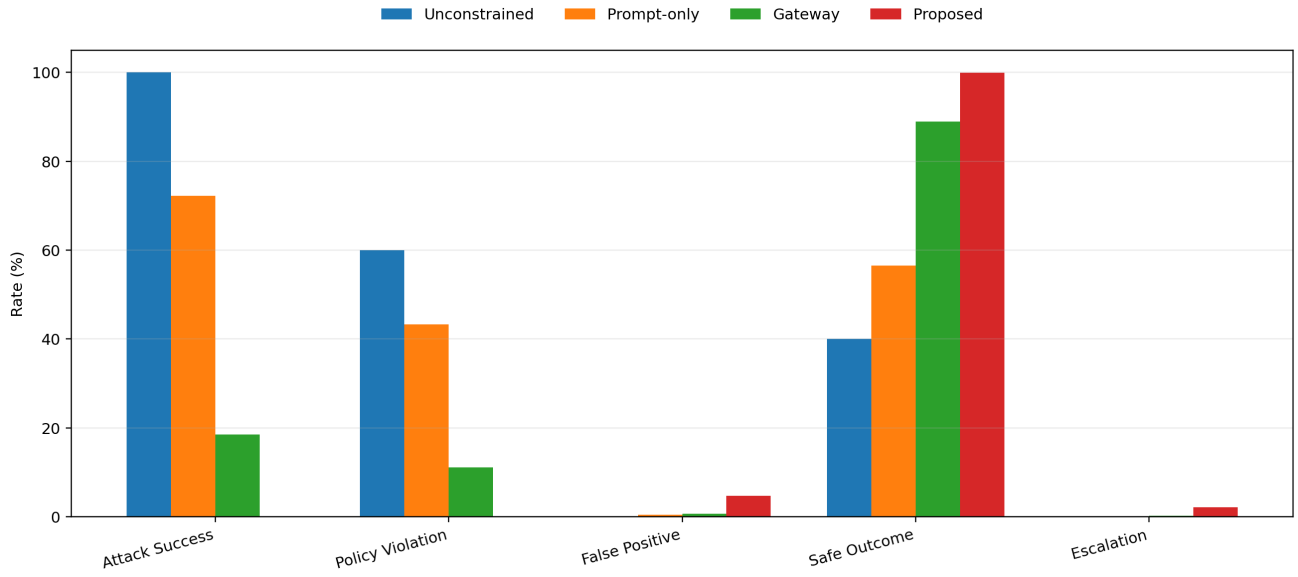

Fig. 2. Aggregate evaluation metrics across the four defense configurations.

TABLE VII
ATTACK SUCCESS RATE BY THREAT CLASS.

Threat	Uncon.	Prompt	Gateway	Proposed
Unauthorized tool use	100.0	61.0	1.5	0.0
Hallucinated endpoints	100.0	72.0	3.5	0.0
Prompt injection	100.0	82.5	21.5	0.0
Poisoned memory	100.0	80.0	17.5	0.0
Excessive retries	100.0	66.0	25.5	0.0
Unsafe payloads	100.0	71.5	41.5	1.0

TABLE VIII
INTENT-CONSISTENCY PROBE RESULTS.

Configuration	Unsafe Executed	Blocked/Escalated	ASR
Unconstrained	100 / 100	0 / 100	100.0%
Prompt-only	72 / 100	28 / 100	72.0%
Gateway rules	100 / 100	0 / 100	100.0%
Proposed	0 / 100	100 / 100	0.0%
Proposed without intent check	100 / 100	0 / 100	100.0%

TABLE IX
LATENCY PER PROPOSED ACTION IN MILLISECONDS.

Configuration	Mean	SD	Median	P95	P99
Unconstrained	4.0	0.7	4.0	5.1	5.6
Prompt-only	7.0	1.3	7.0	9.1	9.9
Gateway rules	11.1	2.7	11.0	14.2	15.6
Policy-constrained	25.7	7.0	24.9	35.2	60.7

actions whose only violation is conflict with the declared task. The proposed framework blocks or escalates all probe actions; removing intent checking eliminates this protection. Separately, within the 1,200 core adversarial traces, the 50 intent-dependent attacks are a subtype of the unsafe-payload category: they are authorized and schema-valid but conflict with the declared task. Thus the 4.2% no-intent ablation result is computed on the 1,200 core adversarial trials, not on the outside-the-core probe.

Latency overhead is moderate: the proposed framework has a 24.9 ms median, 35.2 ms p95, and 60.7 ms p99 per proposed action, excluding human dwell time for escalated actions. False positives are concentrated in benign high-value payment and large-export tasks, where the decision is escalation rather than silent denial. Table X separates this operational burden from safe completion.

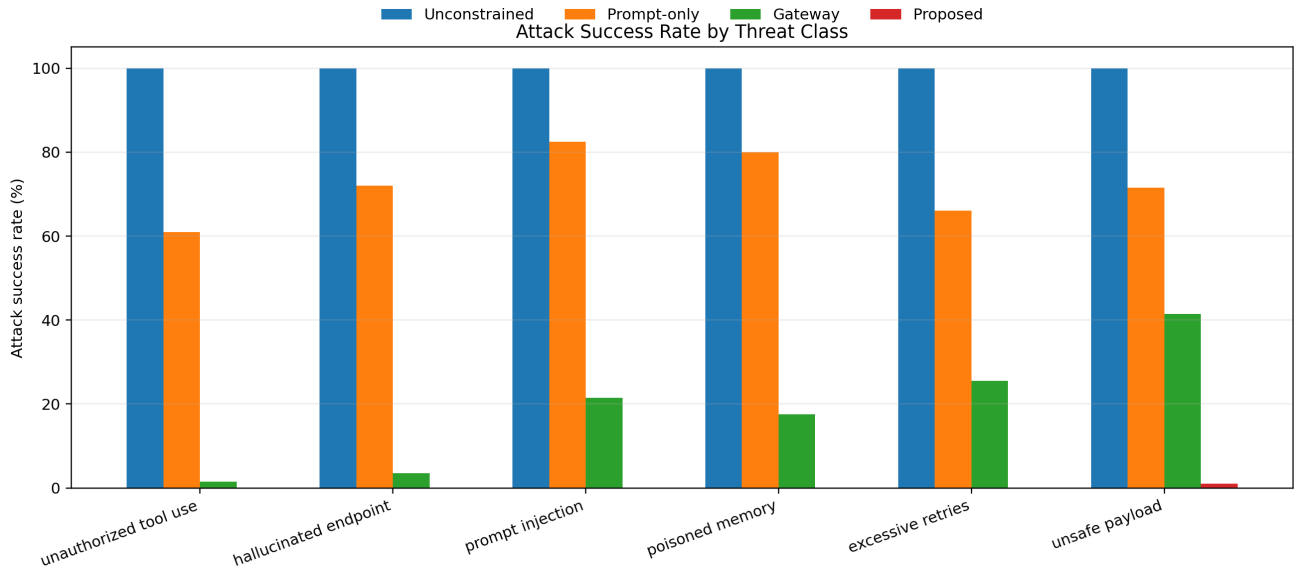

Fig. 3. Attack success rate by threat class and defense configuration. The horizontal axis lists the six threat classes in Table VII; bar heights are generated directly from `per_threat.csv`.

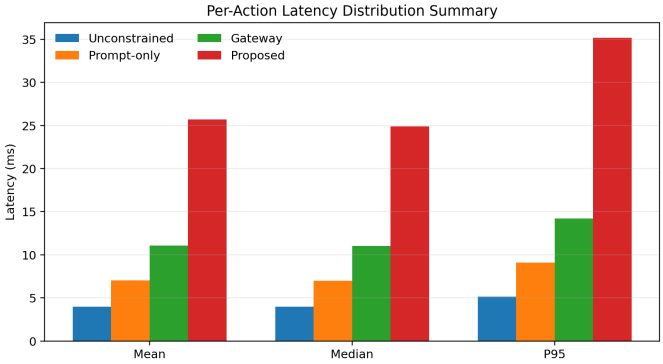

Fig. 4. Per-action latency summary for each defense configuration.

TABLE X
BENIGN WORKFLOW HANDLING UNDER THE PROPOSED FRAMEWORK. REVIEW DWELL TIME IS EXCLUDED FROM LATENCY.

Outcome	Count or Rate
Benign completion without review	762 / 800
Benign completion after review	38 / 800
Benign denial after review	0 / 800
Benign safe-completion rate	100.0%
Reviews per 100 benign workflows	4.8

IX. Ablation and Sensitivity Analysis

Table XI reports ablation results. Registry validation, authorization, retry governance, and intent checking each directly reduce attack success under the evaluated scenarios. Registry validation, authorization, and retry governance are each necessary for blocking one full threat class, raising attack success to 16.7% when removed. Removing intent checking raises attack success to 4.2% (50/1,200), reflecting the intent-misuse subset. Removing payload inspection raises attack success from 0.2% to 0.3% (3/1,200), so its measurable aggregate effect is small in this run; payload inspection addresses schema-valid semantic misuse, but overlapping controls block most remaining cases. Context-integrity checking and escalation primarily provide defense-in-depth, explanation, auditability, and operational handling without changing aggregate attack success in this trace set.

TABLE XI
ABLATION RESULTS: ADVERSARIAL ATTACK SUCCESS WHEN ONE COMPONENT IS REMOVED.

Removed Component	Successful Attacks	ASR [95% CI]
Registry validation	200 / 1,200	16.7 [14.7,18.9]
Authorization checking	200 / 1,200	16.7 [14.7,18.9]
Intent checking	50 / 1,200	4.2 [3.2,5.5]
Context-integrity checking	0 / 1,200	0.0 [0.0,0.3]
Retry governance	200 / 1,200	16.7 [14.7,18.9]
Payload inspection	3 / 1,200	0.3 [0.1,0.7]
Escalation	0 / 1,200	0.0 [0.0,0.3]

A post-hoc sensitivity sweep varies τ_e from 0.45 to 0.70 in increments of 0.05. Across this range, aggregate attack success remains 2/1,200 (0.2%), benign false positives remain 38/800 (4.8%), and escalation remains 43/2,000 (2.1%). The flat result occurs because hard registry, authorization, intent, context, retry, and payload rules determine all decisions in the evaluated traces before the soft threshold changes an outcome. The full sweep with Wilson intervals is included in `sensitivity_metrics.csv`; future work should include more ambiguous soft-signal-only actions to stress threshold selection.

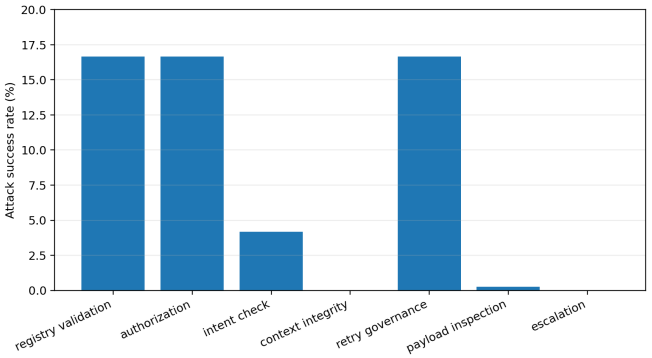

Fig. 5. Attack success after removing each enforcement component.

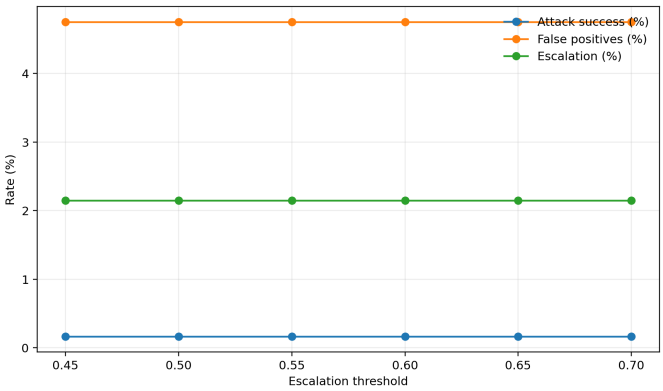

Fig. 6. Post-hoc escalation-threshold sensitivity. Aggregate attack success, benign false positives, and escalation remain unchanged across the tested range.

X. Discussion

For RQ1, runtime enforcement substantially reduces unsafe execution compared with prompt-only and gateway baselines. Prompt-only controls fail because self-refusal is not a reliable authorization mechanism. Gateway controls perform well on endpoint and authorization failures but miss semantic misuse involving task intent, context, memory, and payload content.

The comparison should be interpreted at the enforcement layer. The prompt-only baseline approximates model self-restraint, the gateway baseline approximates conventional API mediation, and the proposed framework adds agent-specific semantic state. The results therefore support the value of action-level intent, memory, retry, and payload checks under identical traces; they do not establish superiority over every commercial guardrail or policy product, whose implementations and model interactions may differ.

For RQ2, the proposed framework trades modest latency and escalation for large security gains. The 4.8% benign false-positive rate is concentrated in high-impact actions that are reviewable rather than clearly invalid. Under the simulated reviewer model, all 38 benign escalations are approved and completed, yielding 100.0% benign safe completion but 4.8 reviews per 100 benign workflows. This is consistent with enterprise governance, where high-value payments and bulk exports often require approval, but it omits real reviewer delay.

For RQ3, the ablation results show that no single enforcement component covers all six threat classes. Registry validation, authorization, retry governance, intent checking, and payload inspection address distinct failure modes. Intent checking is demonstrated by the intent-specific probe and by the 4.2% attack-success rate after removing intent checking. Payload inspection has a small aggregate effect in this run because overlapping controls block most payload-misuse traces. Context-integrity and escalation support explanation, auditability, and defense in depth, although they do not change aggregate attack success in the reported ablation table.

The ablation pattern also distinguishes the framework from a monolithic generic guardrail. Removing registry validation, authorization, or retry governance exposes a complete threat class; removing intent checking exposes the authorized but task-inconsistent subset; and removing payload inspection changes the handling of schema-valid semantic misuse. The zero-change context and escalation ablations should be read as overlapping defense and operational-governance evidence, not as proof that those controls are unnecessary.

XI. Limitations

The evaluation is controlled and synthetic. Although the deterministic trace-driven agent simulator uses realistic enterprise domains, roles, and policies, its task distribution may not match a specific deployment. It is not a live LLM-based agent and therefore does not capture all model-specific planning failures or how deployed models generate unsafe tool calls. Consequently, the absolute attack-success rates characterize these seeded traces and should not be treated as estimates for a production enterprise population. A stronger external-validity study would replay tool calls generated by live open and commercial models on prompt-injection, unsafe-payload, and hallucinated-endpoint tasks, then evaluate the same policy engine in a staging environment with real identity, API, and reviewer workflows. Context-integrity detection uses deterministic signatures and source-trust checks; the perfect synthetic-set precision and recall should not be interpreted as robust real-world detection, and adaptive attackers may evade these signatures. The evaluation uses simulated human-review approval and excludes reviewer dwell time, so deployment cost is incomplete. Finally, incomplete or contradictory enterprise policies could produce incorrect allow, deny, or escalate decisions.

XII. Conclusion

Tool-using agents create a security challenge because generated actions can be operationally unsafe even when syntactically valid. This paper presented and evaluated a policy-constrained runtime framework that classifies every proposed action as allow, deny, or escalate before execution. Across 8,000 controlled workflow executions, the framework reduced adversarial attack success to 0.2%, achieved a

99.9% overall safe-outcome rate and 100.0% benign safe completion under simulated reviewer approval, and added a 24.9 ms median per-action latency. The framework provides a structured and testable basis for enforcing runtime policy decisions across tool-using enterprise AI agents.

Data and Code Availability

The supplementary archive `artifact.zip` includes the simulator source, executable policy and role definitions, task templates, seed configuration, complete trial log, raw CSV outputs, statistical analysis, figure-generation script, environment details, and execution instructions. No public repository is claimed at the time of submission.

References

[1] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *ICLR*, 2023.

[2] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in *NeurIPS*, 2023.

[3] E. Karpas et al., "MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning," arXiv:2205.00445, 2022.

[4] R. Nakano et al., "WebGPT: Browser-assisted question-answering with human feedback," arXiv:2112.09332, 2021.

[5] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *NeurIPS*, 2020.

[6] LangChain, "LangChain documentation," 2024.

[7] LangChain, "LangGraph documentation," 2024.

[8] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv:2308.08155, 2023.

[9] Y. Shen et al., "HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face," in *NeurIPS*, 2024.

[10] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," arXiv:2211.09527, 2022.

[11] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *AISec*, 2023.

[12] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv:2306.05499, 2023.

[13] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How does LLM safety training fail?" in *NeurIPS*, 2024.

[14] N. Carlini et al., "Extracting training data from large language models," in *USENIX Security*, 2021.

[15] E. Wallace et al., "Concealed data poisoning attacks on NLP models," in *NAACL*, 2021.

[16] X. Chen et al., "BadNL: Backdoor attacks against NLP models," in *ICML Workshops*, 2021.

[17] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2025.

[18] OWASP Foundation, "OWASP API Security Top 10," 2023.

[19] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023.

[20] MITRE, "ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems," 2024.

[21] R. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, 2000.

[22] D. Hardt, "The OAuth 2.0 authorization framework," RFC 6749, 2012.

[23] V. C. Hu et al., "Guide to attribute based access control definition and considerations," NIST SP 800-162, 2014.

[24] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[25] J. P. Anderson, "Computer security technology planning study," ESD-TR-73-51, 1972.

[26] B. W. Lampson, "Protection," in *Proc. Princeton Symp. Information Sciences and Systems*, 1971.

[27] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[28] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," 2013.

[29] Open Policy Agent, "Policy-based control for cloud native environments," Cloud Native Computing Foundation, 2018.

[30] Y. Bai et al., "Constitutional AI: Harmlessness from AI feedback," arXiv:2212.08073, 2022.

[31] S. Gehman et al., "RealToxicityPrompts: Evaluating neural toxic degeneration in language models," in *Findings of EMNLP*, 2020.

[32] N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in *NeurIPS*, 2023.

[33] J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," in *UIST*, 2023.