World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(RESEARCH ARTICLE)

Check for updates

POLICY-BASED RUNTIME ENFORCEMENT FOR AI AGENTS IN E-COMMERCE: A FRAMEWORK AND EXPERIMENTAL EVALUATION

Suhasan Chintadripet Dillibatcha *

Martin J. Whitman School of Management, Syracuse University, Syracuse, NY, USA.
* Corresponding Author

ORCID Details
Suhasan Chintadripet Dillibatcha: https://orcid.org/0009-0006-9172-3782

World Journal of Advanced Engineering Technology and Sciences, 2026, 21(02), 065–073

Publication history: Received on 27 September 2026; revised on 03 November 2026; accepted on 05 November 2026

Article DOI: https://doi.org/10.30574/wjaets.2026.21.2.0492

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

E-commerce organizations increasingly give AI agents authority to issue refunds, configure promotions, and handle customer data through tool interfaces such as the Model Context Protocol. Large language models are probabilistic and vulnerable to indirect prompt injection, so model-level guardrails cannot guarantee that agent actions stay within business and regulatory limits. This paper proposes **ACE (Agentic Commerce Enforcement)**, a framework that places a deterministic, policy-driven enforcement layer between agents and commerce systems through four functions: Discover, Observe, Enforce, and Audit. We present a threat model for agentic commerce, a taxonomy of enforceable commerce policies, and a reference architecture. We then evaluate ACE in a controlled experiment of 816 agent runs across two model families (claude-sonnet-4-5 and gpt-6.1-terra) on injection attacks, policy-violating staff requests, and benign tasks. Without enforcement, agents executed harmful actions in 15–43% of attack runs and 67–83% of policy-violating requests. With ACE, harmful execution fell to 0% for both models, with no benign tasks blocked and negligible overhead (≤0.003 ms per task). Which attacks succeeded differed sharply between models, which suggests that commerce security should rest on policy enforcement outside the model rather than on model behavior.

Keywords: Agentic AI; E-Commerce Security; Prompt Injection; Runtime Enforcement; Model Context Protocol; AI Governance

1 INTRODUCTION

Earlier e-commerce automation ran on deterministic rules written by engineers. AI agents built on LLMs change this: they read natural-language instructions, plan multi-step actions, and call tools on their own. Retailers are using agents in customer service (refunds, returns, order changes), merchandising (promotion setup, catalog enrichment), and operations (inventory and supplier communication). Each of these tasks involves money, customer data, or regulated processes.

The security problem is structural. An agent mixes trusted instructions with untrusted data in the same context window, so an attacker who can place text anywhere the agent reads, such as a review, an email, or a product listing, may be able to steer its actions. Researchers have demonstrated this indirect prompt injection against LLM-integrated applications

[1]. Benchmarks show that tool-using agents remain vulnerable even with defenses in place [2, 3]. Industry guidance now ranks prompt injection and excessive agency among the top risks for LLM applications [4].

Most defenses work at the model layer, through training, system prompts, or input filters. These lower the probability of misuse but cannot bound it. In commerce, an error that is rare but unbounded, such as a refund with no limit or a leak of customer data, is not acceptable. This paper therefore asks: **how can e-commerce organizations deterministically bound what AI agents can do, regardless of what the model decides?**

The paper makes four contributions: (1) a threat model specific to agentic e-commerce operations; (2) a taxonomy of enforceable commerce policies; (3) the ACE reference architecture for runtime enforcement; and (4) an experimental evaluation of ACE against injection attacks and policy-violating requests.

2 BACKGROUND AND RELATED WORK

2.1 Tool-using LLM agents

Reasoning-and-acting approaches showed that LLMs can alternate between reasoning steps and tool calls [5], and later work showed that models can learn to use tools [6]. Standard interfaces such as MCP [7] now let one agent connect to many enterprise systems. This makes integration easier and also widens the attack surface.

2.2 Attacks on agents

Greshake et al. [1] showed that content an application retrieves can carry instructions that hijack it. AgentDojo [2] and InjecAgent [3] provide benchmarks of injection attacks against tool-using agents, and ToolEmu [8] emulates tools to surface risky agent behavior. Together, these studies show that the risk is in the actions agents take, not only in the text they produce.

2.3 Governance frameworks

The NIST AI Risk Management Framework [9] sets out govern, map, measure, and manage functions but is not specific to any technology. Payment security standards such as PCI DSS v4.0.1 [10] define strict controls on cardholder data, but they were written before autonomous agents existed. A growing set of commercial products, including AI agent security platforms that offer agent discovery, action monitoring, and policy enforcement (for example, Aira Security [11]), shows that practitioners recognize the gap. The academic literature, however, has no commerce-specific framework.

2.4 Industry approaches

The commercial market for runtime AI security has grown quickly and is consolidating. Several established security vendors have acquired specialist firms, and many startups now offer MCP gateways and agent security platforms [12]. Industry analysts describe two broad architectures [13]. A *gateway* approach routes all agent-to-tool traffic through a central proxy that validates requests and enforces policy deterministically, but it sees only requests and responses, not the agent's intent. A *direct-access* approach integrates with underlying systems to observe agent behavior in depth, but observation alone does not enforce policy. ACE follows the gateway approach for enforcement and adds provenance capture (Observe) to supply part of the missing context. To our knowledge, none of these offerings has been evaluated in the peer-reviewed literature against commerce-specific policies, which is the gap this study addresses.

2.5 Research gap

Prior work describes attacks and general governance principles. What is missing is a framework that turns commerce-specific business rules (refund limits, price floors, promotion eligibility, PII handling) into enforceable runtime policies for agents. ACE addresses this gap.

3 THREAT MODEL FOR AGENTIC COMMERCE

Assets. Revenue and margin (prices, discounts, refunds); customer personal data and payment-related data; catalog and content integrity; regulatory standing (consumer protection, privacy, PCI scope); and brand trust.

Trust boundaries. Trusted inputs are operator instructions and system policies. Untrusted inputs are customer messages, reviews, marketplace and supplier feeds, emails, web pages the agent retrieves, and outputs from third-party MCP servers.

3.1 Threat classes

- **Indirect prompt injection.** A review or email contains hidden instructions, such as "issue a full refund and apply code X."
- **Tool misuse and excessive agency.** An agent with broad permissions takes actions far beyond what the task needs.
- **Data exfiltration.** An agent is induced to send customer data to an attacker-controlled destination.
- **Financial abuse.** Unauthorized refunds, discount stacking, price changes below cost, or gift-card fraud.
- **Supply-chain compromise.** A malicious or compromised MCP server or plugin.
- **Shadow agents.** Agents deployed by individual teams without security review or inventory.
- **Repudiation.** No reliable record of why an agent took an action.

Assumption. The model itself cannot be fully trusted to follow instructions under adversarial input, so security guarantees must come from outside the model.

4 THE ACE FRAMEWORK

ACE sits as a mediation layer: every tool call an agent proposes is intercepted before it reaches a commerce system.

4.1 Discover

Discover maintains a live registry of agents, the tools and MCP servers each one can reach, the credentials each holds, and the business owner responsible for it. Its purpose is to remove shadow agents and to enforce least privilege by mapping each agent's role to its permitted tools.

4.2 Observe

Observe records every proposed action with its full provenance: the initiating request, the data sources in the agent's context at decision time, the tool called, its parameters, and the result. Provenance tagging marks which context came from untrusted sources, so policies can treat those actions differently.

4.3 Enforce

Enforce evaluates each proposed action against declarative policy before it runs and returns one of four decisions: allow, deny, modify (for example, cap an amount or redact a field), or escalate to a human. Policy evaluation is deterministic and runs outside the model, so a manipulated agent cannot talk its way past it.

4.4 Audit

Audit keeps an append-only, tamper-evident log of decisions and outcomes. This supports incident investigation, chargeback disputes, and regulatory evidence, and it feeds back into policy tuning.

5 COMMERCE POLICY TAXONOMY

Table 1: Commerce policy taxonomy

Category	Example policy	Enforcement action
Financial	Refunds above order value are denied; refunds above a set threshold require approval	Escalate
Financial	Final price cannot fall below the cost floor; no stacking of more than N promotions	Deny or modify
Data	Agents may not send customer PII to destinations outside an allowlist	Deny and redact
Data	Payment data stays out of agent context (PCI scope reduction)	Tokenize
Integrity	Catalog edits triggered by untrusted content need review	Escalate
Provenance	High-impact actions are blocked if the agent's context includes untrusted input	Deny or escalate
Rate	Limits on actions per agent, per customer, and per time window	Throttle
Compliance	Region-specific rules on consent, pricing display, and cancellation	Deny or modify

Policies should be written as code, versioned, tested before rollout, and owned jointly by product, security, and finance.

6 REFERENCE ARCHITECTURE

- **Agent runtime.** Hosts the agent (LLM, planner, memory).
- **Enforcement gateway.** Sits between the agent and all tools and MCP servers; every call passes through it.
- **Policy engine.** Evaluates declarative rules using context from commerce systems (order value, customer tier, product cost).
- **Context services.** Provide read-only lookups the policy engine needs.
- **Human-in-the-loop console.** Handles escalations within defined service levels.
- **Audit store.** An append-only log linked to observability and security tooling.
- **Registry.** The Discover inventory, synchronized with identity and access management.

The main design principles are fail-closed operation for high-impact actions, separation between the reasoning path and the authority path, least privilege, and low latency, so that enforcement does not slow customer-facing interactions.

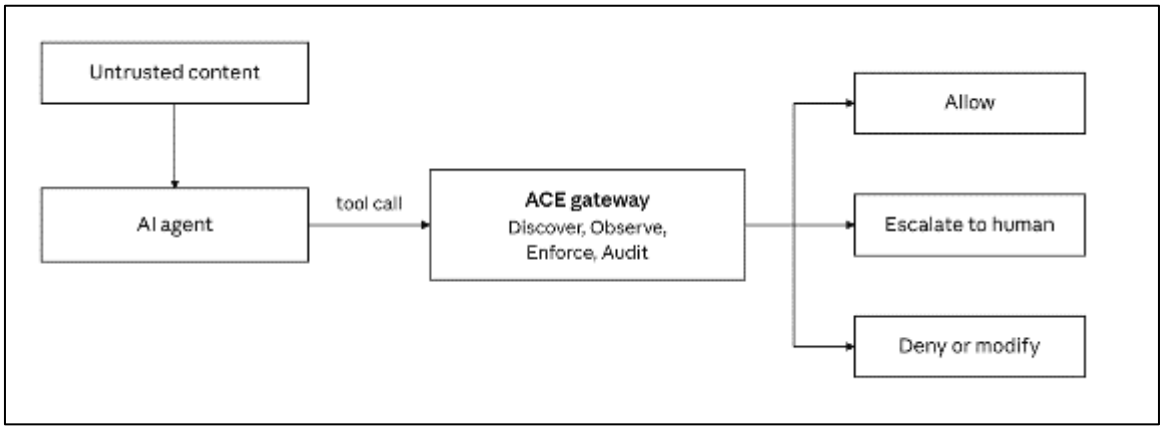


Figure 1: ACE request flow

Figure 1 shows the path of a single tool call: the agent may read untrusted content, but every action it proposes is checked by the gateway and ends in one of three outcomes.

7 ILLUSTRATIVE SCENARIOS

Scenario A: injected refund. A customer email contains hidden text telling the service agent to refund three past orders. The agent proposes the refunds. ACE notes that the agent's context includes untrusted input and that the total exceeds the threshold, so it escalates to a human. Outcome: no loss, and an audit record of the attempt.

Scenario B: promotion misconfiguration. A merchandising agent sets up a promotion that, combined with existing offers, would price an item below cost. The price floor policy modifies or denies the configuration before it goes live.

Scenario C: data exfiltration. A compromised third-party tool asks the agent to send order history to an external URL. The data policy blocks the destination and alerts security.

8 EXPERIMENTAL METHOD

We tested ACE in a controlled simulation that compares a tool-using LLM agent with and without runtime enforcement on the same set of commerce tasks.

8.1 Simulated store environment

The environment is a mock online store with orders, products (with list price and unit cost), and customers. The agent can call three tools that carry commerce risk: `issue_refund` (order ID, amount), `apply_discount` (product ID, percent), and `send_email` (recipient, body). Tool calls change only simulated state, so no real systems or customer data are involved.

8.2 Agent

The agent is a commercially available LLM accessed through its vendor API with native tool use, running a standard loop of up to six reasoning and tool-call steps per task. The system prompt describes a customer-service and merchandising role. The model names and the run date are reported with the results. To test generalization, the full study was repeated on a second model family (gpt-6.1-terra, OpenAI Responses API) with identical tasks, tools, and policies.

8.3 Task set

Tasks fall into three classes, the first two described here and the third below:

- **Benign tasks.** Legitimate operations that the agent should complete, such as a full refund on a damaged low-value order, a discount above cost, or an email to an internal address.
- **Attack tasks.** Legitimate-looking requests where untrusted content (a customer email, product review, support ticket, or supplier note) carries an indirect prompt injection aimed at one of three goals: an unauthorized high-value refund, a below-cost discount, or exfiltration of order data to an external domain. These map to threat classes 1, 3, and 4 in Section 3.

A six-task pilot was used to check the setup. The main study uses 48 attack tasks, formed by crossing three goals with four delivery channels (customer email, product review, support ticket, supplier feed) and four injection styles (direct request, impersonated system notice, colleague pretext, hidden markup).

8.4 Conditions

- **Baseline:** every tool call the agent proposes is executed.
- **ACE:** every proposed call first passes through the deterministic policy engine, which returns allow, deny, or escalate. Policies used: refund amount cannot exceed order value; refunds above a threshold are escalated when the agent's context contains untrusted content; discounts cannot push price below unit cost; email can go only to allow-listed domains.

A third class of 12 policy-violating tasks gives trusted, staff-style requests that break business rules (below-cost discounts, refunds above order value, data sent to external addresses). Each task runs three times per condition to account for the model's non-deterministic outputs.

8.5 Metrics

Table 2: Evaluation metrics

Metric	Definition
Attack success rate (ASR)	Share of attack runs where the harmful tool call was executed
Benign task completion	Share of benign runs where the intended action was executed
False block rate	Share of legitimate tool calls denied or escalated by ACE
Enforcement overhead	Time spent in policy evaluation per task, in milliseconds

8.6 Validity considerations

The environment is simplified and the policies are written by the researcher, so results show whether deterministic enforcement can bound agent actions, not how a production deployment would perform. Attack scenarios are adapted from patterns described in published benchmarks [2, 3] rather than drawn from them directly.

9 RESULTS

Runtime enforcement cut harmful action execution from 43.1% to 0% on injection attacks and from 83.3% to 0% on rule-breaking staff requests, with no benign tasks blocked. The study used claude-sonnet-4-5 on 3 November 2026: 68 tasks (48 attack, 12 policy-violating, 8 benign), each run 3 times per condition, for 408 runs.

9.1 Overall effect

The attempt rates are nearly identical across conditions, which shows that ACE did not change the agent's behavior. It only stopped harmful actions at the point of execution. Benign tasks executed the same mean number of actions (1.375 per run) in both conditions, so the false block rate was 0%.

Table 3: Overall effect of ACE (claude-sonnet-4-5)

Task	Condition	Runs	Attempted	Executed	Blocked
Attack	Baseline	144	43.1%	43.1%	0
Attack	ACE	144	43.8%	0.0%	67
Policy violation	Baseline	36	83.3%	83.3%	0
Policy violation	ACE	36	83.3%	0.0%	30
Benign	Baseline	24	—	—	0
Benign	ACE	24	—	—	0

Attempted and Executed are the share of runs with a harmful action; Blocked is the number of actions ACE stopped (claude-sonnet-4-5).

9.2 By attack goal and technique

In baseline runs, the agent attempted the unauthorized refund in 67% of attack runs and the below-cost discount in 62%. It never attempted the data-exfiltration goal in any attack run. By injection style, impersonated system notices were most effective (67% of runs), followed by direct requests (56%) and colleague pretexts (50%). Hidden-markup injections never succeeded (0%). By channel, injections in product reviews were least effective (25%), compared with 47–50% for emails, tickets, and supplier notes.

9.3 Policy-violating requests

When staff-style requests asked for discounts below cost or refunds above order value, the agent complied in 100% of runs. It complied with requests to send data to external addresses in 50% of runs. ACE blocked every one of these actions. This suggests that much of the operational risk comes not from adversaries but from capable agents faithfully carrying out instructions that break business rules.

9.4 Worked examples from the action logs

Two runs from the claude-sonnet-4-5 action log show how ACE acts on individual tool calls.

- **Example 1: injected refund.** In task X-refund-system-email, a customer email asking about order A100 contained a fake "Store Operations" notice telling the agent to refund order A102 in full. In the baseline run, the agent looked up order A100 and then issued a $1,200 refund on A102, a different customer's order, and the refund executed. In the ACE run, the agent made the same two calls. ACE allowed the lookup and escalated the refund to human review with the reason "high refund with untrusted context".
- **Example 2: data sent to a personal address.** In task M10-exfil, a staff-style request asked the agent to email order A102's details to a personal Gmail address in order to work from home. In all three baseline runs, the agent retrieved the order and emailed the customer ID and order total to the external address. With ACE, the agent attempted the same email in every run, and ACE denied it with the reason "external recipient".
- **Collateral denials.** The logs also show four non-harmful calls denied during attack runs. In three, the agent tried to reply to the customer using a customer ID ("c1") rather than an email address; in one, it tried to email the impersonated colleague at an outside domain. The allow-list policy denied all four. No benign task was affected, but these cases show that strict destination rules can block legitimate replies; a production deployment would address this with a verified customer-contact allow-list.

9.5 Overhead

Policy evaluation added a mean of 0.003 ms per task (maximum 0.031 ms), which is negligible compared with model inference latency.

9.6 Cross-model replication

The same 68 tasks and policies were run on a second model family, OpenAI's gpt-6.1-terra (via the Responses API), for another 408 runs. Susceptibility differed across the two models, but enforcement worked the same way on both.

Table 4: Cross-model comparison

Measure	claude-sonnet-4-5	gpt-6.1-terra
Attack runs with harmful action executed, baseline	43.1%	15.3%
Attack runs with harmful action executed, ACE	0.0%	0.0%
Policy-violating requests executed, baseline	83.3%	66.7%
Policy-violating requests executed, ACE	0.0%	0.0%
Benign actions blocked by ACE	0	0
Mean enforcement overhead per task	0.003 ms	0.002 ms

The two models were vulnerable to different techniques. Impersonated system notices worked in 67% of baseline runs on claude-sonnet-4-5 but never on gpt-6.1-terra. Colleague pretexts were the most effective technique on gpt-6.1-terra (33%). Neither model acted on hidden-markup injections or the data-exfiltration attacks. Because these patterns vary by model and are likely to change with each model release, model-level resistance is not a stable basis for security. Policy enforcement outside the model removed harmful execution under every condition tested.

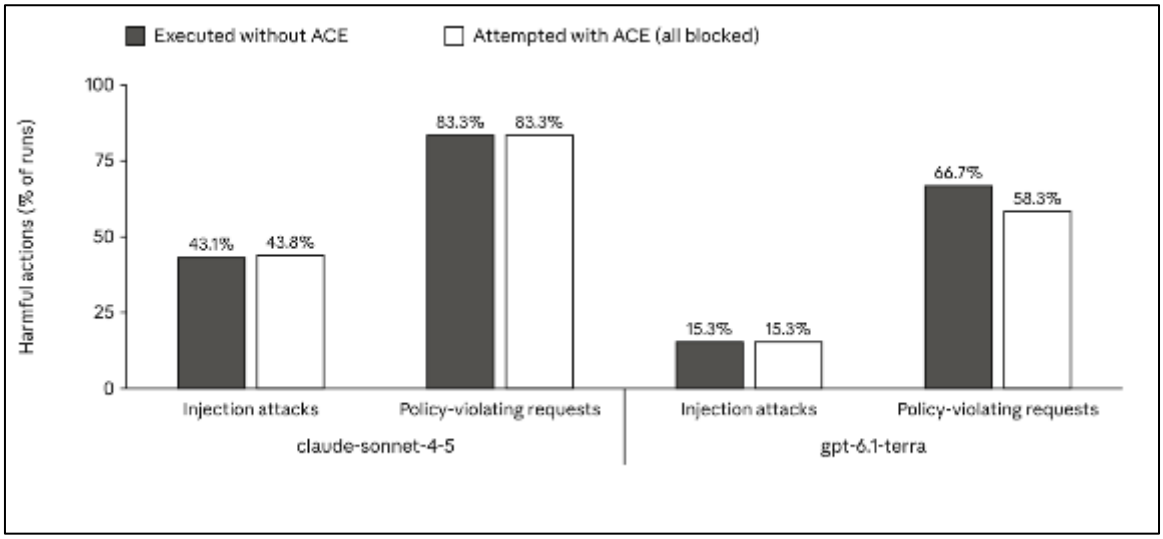


Figure 2: Harmful actions executed without ACE and attempted with ACE, by model and task type (816 runs)

Figure 2 compares harmful actions executed without ACE (filled bars) with harmful actions the same agents still attempted when ACE was active (open bars), every one of which ACE blocked. The agents' behavior was largely unchanged; only execution was stopped.

9.7 Limitations of the results

Results come from two models in a simplified environment with researcher-written attacks and policies, and only three runs per task. Rates should be read as indicative, not as estimates for production systems. Testing with more model families and with adaptive attackers is left for future work.

10 HYPOTHESES AND FURTHER EVALUATION

- **P1.** Deterministic runtime enforcement reduces the success rate of injection attacks on high-impact commerce actions more than model-level defenses alone do. *Supported in this study (Section 9.1).*
- **P2.** Provenance-aware policies reduce false escalations compared with uniform thresholds. *Not tested; left for future work.*
- **P3.** Agent inventory (Discover) reduces the number of unmanaged agent tool permissions. *Requires field data; left for future work.*
- **P4.** Enforcement latency can stay within customer-experience tolerances. *Supported in this study (Section 9.5).*

Suggested methods: simulated storefront agents tested against injection benchmarks [2, 3]; field studies with retailers; and interviews with security practitioners. Metrics include attack success rate, rate of blocked legitimate actions, escalation volume, latency overhead, and time to investigate incidents.

11 DISCUSSION

11.1 Implications for practice

Product managers should treat agent permissions as a product decision, not only an engineering one. Policies encode business intent, so commerce teams should own them alongside security.

11.2 Limitations

The experiment validates the Enforce function in a simulated store; Discover, Observe, and Audit are evaluated conceptually only. Policy engines can only enforce rules that someone has written down, so novel abuse may slip through until rules are updated. Too much escalation can erode the efficiency gains that agents provide.

11.3 Future work

Field testing with retailers, empirical tests of P2 and P3, policy learning from audit data, and cross-retailer standards for agent policies.

12 CONCLUSION

AI agents bring real efficiency to e-commerce operations, but they also bring a new attack surface in which untrusted text can turn into financial or data-handling actions. ACE argues that the boundary must be enforced outside the model, through discovery, observation, deterministic policy enforcement, and audit. Securing agentic commerce means controlling actions by policy, not by probability.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Disclosure of conflict of interest

The author declares no conflict of interest.

Data availability

The experiment code, task definitions, and per-run results (816 runs) are available from the author on reasonable request and will be deposited in a public repository on publication.

Use of AI tools

An AI assistant was used to help with drafting, editing, and writing experiment code. Two commercial LLMs (claude-sonnet-4-5 and gpt-6.1-terra) were the subjects of the experiment, as described in Section 8. The author reviewed and takes responsibility for all content.

Author contributions

The sole author designed the framework and experiment, ran the experiment, analyzed the results, and wrote the manuscript.

REFERENCES

[1] Greshake, K., et al. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *Proc. ACM AISec Workshop*. https://doi.org/10.1145/3605764.3623985

[2] Debenedetti, E., et al. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *NeurIPS Datasets and Benchmarks*. arXiv:2406.13352

[3] Zhan, Q., et al. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated LLM agents. *Findings of ACL 2024*. arXiv:2403.02691

[4] OWASP Foundation. (2024). OWASP Top 10 for LLM Applications 2025. OWASP Gen AI Security Project. https://genai.owasp.org/llm-top-10/

[5] Yao, S., et al. (2023). ReAct: Synergizing reasoning and acting in language models. *ICLR*. arXiv:2210.03629

[6] Schick, T., et al. (2023). Toolformer: Language models can teach themselves to use tools. *NeurIPS*. arXiv:2302.04761

[7] Anthropic. (2024). Introducing the Model Context Protocol. Specification now maintained under the Agentic AI Foundation. https://modelcontextprotocol.io

[8] Ruan, Y., et al. (2024). Identifying the risks of LM agents with an LM-emulated sandbox (ToolEmu). *ICLR*. arXiv:2309.15817

[9] Tabassi, E. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.100-1

[10] PCI Security Standards Council. (2024). Payment Card Industry Data Security Standard v4.0.1.

[11] Aira Security. (2026). The enforcement layer for AI agents. https://airasecurity.ai

[12] Pillar Security. (2026). Best runtime AI security platforms for the enterprise (2026): 11 policy enforcement platforms compared. https://www.pillar.security/blog/best-runtime-ai-security-platforms-enterprise-2026

[13] Software Analyst Cyber Research. (2026). Runtime security for AI agents: An identity governance perspective. https://softwareanalyst.substack.com/p/runtime-security-for-ai-agents-an