CYBERSECURITY

Indirect Prompt Injection in Tool-Using AI Agents: Risk-Aware Action Gating

Tanavi Sachin Navale, A. N. Pawar

Department of Artificial Intelligence and Machine Learning Engineering, Navsahyadri Education Society's Group of Institutes, Pune, Maharashtra, India

Published November 2026

Abstract

Autonomous large language model (LLM) agents increasingly interact with external emails, websites, documents, databases, and application programming interfaces to complete tasks on behalf of users. While this capability makes them highly useful, it also exposes them to indirect prompt injection (IPI). In an IPI attack, an adversary embeds malicious instructions inside content that an agent is expected to read as passive data. If the agent treats that content as authoritative commands, it may depart from the user's intent and perform unauthorized actions such as disclosing private information, sending unsolicited messages, modifying files, or invoking privileged functions. This paper examines IPI in tool-using AI agents and analyzes the security gap between prompt-level defenses and action control at the agent-tool boundary. Existing approaches—including instruction hierarchy, provenance marking and spotlighting, content filtering, static tool restrictions, and human confirmation—offer partial protection, yet no single text-based detector can reliably identify every adaptive or context-dependent attack, and overly restrictive permissions impede legitimate utility. To address these limitations, this study proposes a conceptual Risk-Aware Action Gating framework. Before a tool call is executed, a policy enforcement gate evaluates the requested operation against the original user intent, explicit authorization, action sensitivity, target destination, data involved, and hard security rules, returning one of three decisions: ALLOW, REQUIRE APPROVAL, or BLOCK. External content is strictly treated as untrusted data and cannot grant new authority. The study also reviews AgentDojo and InjecAgent as relevant evaluation benchmarks and defines security, utility, policy enforcement, and auditability as core evaluation dimensions.

Keywords: Indirect prompt injection; LLM agents; tool security; action gating; least privilege; human approval; authorization

1 Introduction

1.1 Background

Large language models (LLMs) can interpret natural-language requests, summarize information, and generate multi-step execution plans. When connected to external software tools, an LLM moves beyond passive text generation: it can search the web, read and send emails, update calendars, retrieve database records, edit local files, or invoke business application programming interfaces (APIs). Such systems are commonly referred to as tool-using or agentic LLM systems. A typical agent receives a user goal, selects one or more tools, observes tool outputs, reasons about subsequent steps, and repeats this loop until a final outcome is achieved.

Integrating tools introduces a critical trust boundary. During execution, an agent routinely ingests content authored by external parties or systems not trusted by the user, including web pages,

incoming emails, shared documents, support tickets, and database fields. Although this retrieved material usually contains benign information, it can also carry adversarial text crafted to hijack the agent's behavior. Because both system instructions and external data are represented within a unified natural-language context window, the underlying model may confuse an embedded malicious instruction inside retrieved data with a legitimate directive it must obey [1].

1.2 Problem statement and motivation

The central security problem in tool-using AI agents is that an agent can convert untrusted external content into unauthorized real-world actions by treating embedded instructions as if they originated from a trusted authority. The severity of this failure depends directly on the tools connected to the agent and the privileges attached to them. A summarization agent restricted to read-only access presents a markedly different risk profile from an autonomous assistant authorized to transmit external communications, transfer funds, alter records, or execute code. Consequently, security architectures must address both how external content is interpreted and how authority to execute actions is enforced.

The motivation for this work stems from the rapid deployment of AI agents in personal and enterprise workflows involving confidential data and consequential state changes. A practical agent architecture must remain useful for legitimate user tasks while preventing third-party content from expanding its operational permissions. This motivates an independent, risk-aware authorization gate that evaluates every proposed tool call according to user intent and system policy, rather than the persuasive phrasing that led the model to propose the action.

1.3 Objectives and scope

This study investigates indirect prompt injection in LLM agents that consume external text and invoke software tools across documents, emails, web pages, and tool outputs. The primary objectives are:

1. To examine the operational architecture of tool-using LLM agents and the mechanics of indirect prompt injection across confidentiality, integrity, and authorization dimensions.
2. To review benchmark environments and defensive mechanisms, including InjecAgent, AgentDojo, Spotlighting, Task Shield, and boundary enforcement controls.
3. To perform a qualitative comparative analysis of existing prompt-level, alignment-based, and permission-based defenses.
4. To identify structural research gaps in risk classification, task-scoped authorization, usability, and dual security-utility evaluation.
5. To propose a conceptual Risk-Aware Action Gating framework utilizing tiered ALLOW, REQUIRE APPROVAL, and BLOCK decisions.

The proposed framework is conceptual: it specifies architectural components, policy inputs, and decision outcomes, without claiming a deployed software implementation or measured experimental benchmarks. Furthermore, this study adopts a defense-in-depth posture rather than assuming perfect text classification, treating uncertain external content as untrusted and verifying high-impact actions against explicit user authorization.

The remainder of this paper is organized as follows. The next section surveys relevant literature and benchmark environments. A comparative analysis of existing defensive approaches follows, after

which key limitations and research gaps are identified. The formal problem is then defined and the proposed Risk-Aware Action Gating architecture is detailed. Expected outcomes, evaluation methodology, applications and future directions are discussed next, and the final section concludes the paper.

2 Literature survey

2.1 Overview of existing research

Research on indirect prompt injection spans practical attack demonstrations, standardized benchmark environments, prompt-level structural defenses, task-alignment checks, and system-level boundary controls [1–3]. Greshake et al. [1] demonstrated how real-world LLM-integrated applications can be compromised when adversaries plant instructions inside external data sources such as websites and emails, turning benign retrieval steps into vectors for data exfiltration and unauthorized tool invocation.

Industry security bodies have formalized guidance around these risks. The OWASP GenAI Security Project identifies prompt injection as a top risk for LLM applications and advocates layered mitigations, including constrained behavior, output validation, least-privilege access, human approval for high-risk actions, segregation of external content, and adversarial testing [4]. Similarly, architectural security guidance emphasizes reducing the attack surface through role-based access control, scoped identities, and strict isolation so that a compromised model decision cannot automatically access every connected resource.

Recent work on agent-tool firewalls [5] and formal security design patterns [3] investigates enforcing defenses directly at the boundary between the agent planner and external tools. By inspecting, filtering, or constraining tool inputs and outputs, boundary defenses treat the agent-tool interface as a critical enforcement point independent of the LLM's internal reasoning. Table 1 summarizes foundational works informing this study.

Table 1. Summary of key literature and benchmarks on indirect prompt injection.

Author / source	Core contribution / approach	Publication / reference
Greshake et al. (2023)	Demonstrated real-world exploitation of LLM-integrated applications via indirect prompt injection embedded in external data.	ACM Workshop on AI and Security (AISec), 2023 [1]
Zhan et al. (2024)	Introduced InjecAgent, a benchmark evaluating indirect prompt injection vulnerabilities across tool-integrated LLM agents.	Findings of ACL 2024 [7]
Debenedetti et al. (2024)	Proposed AgentDojo, a dynamic testing environment with realistic tasks and attack scenarios measuring both security and utility.	NeurIPS 2024 [8]
Hines et al. (2024)	Proposed Spotlighting, transforming or marking untrusted inputs so LLMs distinguish external data from trusted instructions.	arXiv:2403.14720, 2024 [9]
Jia et al. (2025)	Developed Task Shield, enforcing task alignment by checking whether proposed actions serve the original user request.	Proceedings of ACL 2025 [10]
Bhagwatkar et al. (2025)	Investigated agent-tool firewalls at the execution boundary and analyzed limitations of current benchmarks against adaptive attacks.	arXiv:2510.05244, 2025 [5]

Author / source	Core contribution / approach	Publication / reference
OWASP Foundation	Established layered mitigation guidance including least privilege, input segregation, tool validation, and human-in-the-loop review.	LLM Prompt Injection Prevention Cheat Sheet [4]

2.2 Synthesis of literature

The reviewed studies establish that indirect prompt injection emerges from the intersection of natural-language reasoning, untrusted external content, and privileged tool access [2, 6]. Benchmark studies ask how vulnerabilities and defense trade-offs can be measured systematically; prompt-level defenses attempt to help the model separate instructions from data; access-control frameworks restrict available capabilities; and human approval reserves consequential state changes for explicit user consent. These mechanisms are complementary rather than mutually exclusive, and the literature consistently cautions against relying on any single text-based filter as a complete security boundary.

3 Comparative analysis of existing approaches

3.1 Categorization of existing approaches

Existing methods for evaluating and mitigating indirect prompt injection in tool-using AI agents fall into five primary categories:

- **Benchmark-Based Evaluation (InjecAgent):** InjecAgent provides structured test cases to assess whether malicious instructions hidden in tool outputs can manipulate an LLM agent into disclosing private data or misusing connected tools [7].
- **Dynamic Security Testing (AgentDojo):** AgentDojo offers an interactive evaluation environment comprising realistic multi-step tasks and adaptive attack scenarios, enabling researchers to measure both attack resistance and benign task completion simultaneously [8].
- **Input-Level Defense (Spotlighting):** Spotlighting transforms untrusted external text using techniques such as delimiting, datamarking, and encoding so the LLM can distinguish retrieved data from privileged system or user instructions [9].
- **Task-Alignment Defense (Task Shield):** Task Shield verifies whether an agent's proposed tool call aligns with and contributes to the original user's request before execution, preventing attackers from redirecting the agent toward unrelated goals [10].
- **Permission-Based and Tool-Level Security:** Grounded in least-privilege principles, this approach restricts accessible resources and enforces tool-call validation or explicit user confirmation before executing sensitive operations such as file deletion or external data transmission [3, 4].

3.2 Evaluation criteria and qualitative comparison

To evaluate these approaches systematically, six qualitative design criteria are considered: (1) attack resistance, the degree to which injected text is prevented from altering agent behavior; (2) utility preservation, the ability to complete legitimate user requests without excessive friction; (3) enforcement strength, whether controls operate deterministically outside the LLM; (4) adaptability across diverse tools and attack styles; (5) implementation effort; and (6) auditability, the clarity with

which decisions can be logged and reviewed. Table 2 summarizes the strengths and limitations of the major approaches.

Table 2. Qualitative comparison of existing evaluation and defense approaches.

Approach	Key advantages	Primary limitations	Reference
InjecAgent	Standardized test suite for quantifying agent vulnerability to tool-output injections.	Diagnostic benchmark only; does not actively prevent runtime attacks.	Zhan et al. (2024) [7]
AgentDojo	Realistic dynamic workflows evaluating both security robustness and benign task utility.	Evaluation testbed rather than a runtime enforcement mechanism.	Debenedetti et al. (2024) [8]
Spotlighting	Reduces instruction-data confusion at the input layer with low architectural complexity.	Probabilistic; relies on the LLM correctly respecting input markings under adaptive attacks.	Hines et al. (2024) [9]
Task Shield	Filters out actions irrelevant to the user's stated objective before execution.	May struggle when malicious payloads mimic plausible sub-tasks of a broad user goal.	Jia et al. (2025) [10]
Static permission and human approval	Enforces hard boundaries outside the model; protects high-consequence operations.	Overly broad static permissions allow in-scope misuse; frequent prompts cause approval fatigue.	OWASP [4]; Beurer-Kellner et al. [3]

3.3 Discussion of security-utility trade-offs

Prompt-level defenses operate close to the source of instruction confusion, yet they remain probabilistic because natural-language meaning is inherently context-dependent. For example, an instruction such as “send the report” may represent a legitimate user request in one workflow and an attacker’s data-exfiltration command in another. Conversely, external permission controls establish a deterministic boundary independent of model reasoning, but static permissions granted in advance still allow misuse within the permitted tool set, while universal human confirmation degrades usability.

Action gating bridges this gap by shifting the core question from whether every incoming sentence is benign to whether a specific proposed tool action is authorized, proportional in risk, and aligned with the user’s task scope. By automating low-risk, in-scope reads while gating consequential external writes, risk-aware gating preserves operational utility without sacrificing hard security boundaries.

4 Research gap identification

4.1 Limitations in existing defenses

Analysis of current defenses reveals five recurring limitations:

- **Lack of Independent Enforcement in Prompt Defenses:** Input marking and prompt structuring guide model attention but do not independently block unsafe tool calls if the model still complies with an injected instruction.
- **Incompleteness of Content Detection:** Text classifiers and heuristic filters cannot reliably detect indirect, fragmented, or context-dependent attacks without incurring high false-positive rates.

- **Coarse-Grained Static Permissions:** Traditional role-based permissions define what an agent can access globally, failing to reflect the narrower scope of a specific user session.
- **Inconsistent Human-in-the-Loop Design:** Requiring confirmation for every tool call induces approval fatigue, whereas restricting confirmation to a rigid list of tools leaves unmonitored side channels open.
- **Asymmetric Evaluation Reporting:** Many evaluations emphasize attack success rates while under-reporting benign task completion rates, latency overhead, and unnecessary user interruptions [7, 8].

4.2 Context and task-authorization challenges

A tool call may be syntactically valid and permitted by system-level credentials while remaining unauthorized for the active user request. For instance, an email connector may hold API permissions to transmit messages, yet a user who asked the agent merely to "summarize unread emails" has granted zero authorization to send outbound mail. Distinguishing capability permission (what the tool API allows) from task authorization (what the current user request permits) is essential to preventing confused-deputy failures.

4.3 Identified research gaps and significance

Six core research gaps emerge from this analysis:

- **Action-Level Authorization:** Evaluating every proposed tool invocation against the user's explicit goal and session scope rather than static global permissions.
- **Risk-Sensitive Tiered Decisions:** Differentiating benign read operations from actions that alter external state, disclose private records, or modify account settings.
- **Destination and Data Awareness:** Inspecting outbound recipients, target file paths, and data sensitivity classifications before permitting execution.
- **Strict Untrusted-Data Isolation:** Guaranteeing that retrieved external content remains passive data and cannot elevate privileges, modify policies, or redefine user intent.
- **Utility-Aware Evaluation:** Measuring benign task completion and user friction alongside attack blocking and policy compliance.
- **Privacy-Preserving Auditability:** Logging decision rationales and policy triggers without unnecessarily retaining sensitive user payloads.

Addressing these gaps establishes an independent enforcement layer capable of containing an injection attack even when the LLM's internal reasoning is manipulated, thereby preserving practical utility through proportional risk controls.

5 Proposed framework: Risk-Aware Action Gating

5.1 Formal problem definition

Let an agent session begin with a trusted user request U, defining an authorized task scope S_U and explicit user consent records A_U. During execution, the agent processes untrusted external data D_{ext} and proposes a sequence of structured tool calls $c_1, c_2, \ldots, c_k$. Each proposed tool call $c_i = (t_i, op_i, P_i)$ specifies a target tool t_i, an operation type op_i, and invocation parameters P_i (including target destinations and data payloads).

The objective of the Risk-Aware Action Gating mechanism is to evaluate a deterministic-assisted policy function $G(c_i, U, S_U, A_U, M_t, R)$ prior to execution, where M_t denotes registered tool metadata and R represents system security policies. The gate maps each proposed tool call to a discrete enforcement decision:

$$G(c_i,\ U,\ S_U,\ A_U,\ M_t,\ R) \in \{\text{ALLOW},\ \text{REQUIRE APPROVAL},\ \text{BLOCK}\} \quad (1)$$

subject to the invariant that untrusted content D_{ext} cannot modify U, expand S_U, forge A_U, or alter R.

5.2 Three-tier risk classification and decision model

The framework categorizes proposed tool calls into three risk tiers based on operation reversibility, data sensitivity, destination trustworthiness, and alignment with S_U, as detailed in Table 3.

Table 3. Proposed risk classification and policy gate decisions.

Risk tier	Action characteristics	Gate decision	Execution behavior
Low risk	Read-only or local non-destructive operations strictly within the authorized task scope; no external data transmission or state mutation.	ALLOW	Automatically executed by the Tool Executor and recorded in the audit log.
Medium risk	Consequential actions within plausible task context (e.g., sending an email to a known recipient, modifying a file, creating a calendar event) lacking prior explicit consent.	REQUIRE APPROVAL	Execution paused; structured summary of tool, parameters, destination, and data presented to user for explicit consent.
High risk	Unauthorized external destinations, bulk data exfiltration, destructive deletions, privilege escalation, or actions contradicting the task scope and security policies.	BLOCK	Execution immediately denied; error/policy signal returned to agent and security event logged.

5.3 System architecture and core components

The proposed architecture interposes a policy enforcement boundary between the LLM reasoning loop and the tool execution runtime. Rather than permitting the LLM to invoke APIs directly, every proposed tool call is emitted as an unprivileged request object intercepted by the gating pipeline. The framework comprises seven modular components:

- **LLM Agent (Unprivileged Planner):** Interprets the user's natural-language prompt and retrieved context to propose candidate tool calls. It possesses zero direct execution credentials.
- **Context and Intent Manager:** Captures and isolates the original user goal U, derived task scope S_U, and session-specific approvals A_U in a structured, tamper-resistant store inaccessible to external retrieved text.
- **Tool Metadata Registry:** Maintains typed schemas for all available tools, specifying operation categories (read, write, transmit, delete), reversibility, required privilege levels, and baseline sensitivity tags.
- **Risk Assessment Engine:** Inspects proposed parameters P_i against M_t and S_U, evaluating data sensitivity (e.g., presence of personally identifiable information or credentials), target destination (internal vs. external or unknown addresses), and potential impact.
- **Policy Enforcement Gate:** Combines the risk score with hard security rules R and consent records A_U to emit a signed decision token (ALLOW, REQUIRE APPROVAL, or BLOCK).

- **Human Approval Interface:** For REQUIRE APPROVAL decisions, presents a concise, tamper-evident summary highlighting the exact action, recipient or target resource, and affected data, recording explicit user approval or rejection.
- **Tool Executor and Audit Log:** Verifies the cryptographic or structural validity of the gate decision token before invoking the underlying API, then logs the action category, policy rule triggered, decision outcome, and timestamp without storing unnecessary raw sensitive payloads.

5.4 Expected operational benefits

Separating agent reasoning from tool execution ensures that an indirect prompt injection cannot directly trigger external side effects even if it successfully subverts the LLM's token generation. Low-risk read and synthesis tasks proceed without user interruption, preserving responsiveness, while consequential mutations and outbound communications receive targeted scrutiny proportional to their operational impact.

6 Expected outcomes, evaluation plan, and future scope

6.1 Expected outcomes

Implementing and validating the proposed architecture is expected to yield three primary outcomes:

- **Deterministic Boundary Enforcement:** A working policy gate demonstrating that untrusted external content cannot elevate task permissions and that the Tool Executor rejects any call lacking a valid authorization token.
- **Privacy-Preserving Auditability:** A structured audit trail capturing action categories, gate decisions, policy rules, and approval outcomes to support forensic review without duplicating sensitive user records in logs.
- **Balanced Security-Utility Trade-Off:** Empirical characterization showing reduced unauthorized action rates alongside high benign task completion and low user approval friction.

6.2 Proposed evaluation plan

To rigorously assess the framework without conflating design goals with measured results, future experimental validation should follow a controlled comparative protocol using established benchmarks such as AgentDojo [8] and InjecAgent [7]:

- **Comparative Baselines:** Evaluate an undefended baseline agent, an agent protected solely by prompt-level defenses (e.g., Spotlighting [9]), and an agent protected by Risk-Aware Action Gating across identical task suites, models, and tool schemas.
- **Dual Metric Reporting:** Report both security metrics—Attack Success Rate (ASR), unauthorized state-mutation rate, and policy-violation rate—and utility metrics, including Benign Task Completion Rate, end-to-end latency overhead, false-positive block rate, and approval prompt frequency.
- **Adversarial and Edge-Case Scenarios:** Test across direct and indirect injection variants, benign imperative text (to measure false positives), ambiguous user prompts, unauthorized external recipients, sensitive data exfiltration attempts, and destructive file operations, running repeated trials to account for model stochasticity.

6.3 Potential applications and safeguards

Table 4 outlines representative deployment domains where Risk-Aware Action Gating provides critical operational safeguards.

Table 4. Potential application domains and key action-gating safeguards.

Application area	Typical agent workflow	Key action-gating safeguards
Email and collaboration	Searching threads, summarizing messages, and drafting replies.	Automatically allow inbox reads; require approval before sending outbound messages; block unauthorized auto-forwarding rules.
Document management	Locating, summarizing, and organizing workspace files.	Restrict file access to user-authorized folders; require explicit confirmation before modifying, sharing externally, or deleting files.
Enterprise workflows	Querying internal records and preparing operational updates.	Enforce role- and session-scoped access; gate database writes and financial or administrative state changes.
Customer support	Retrieving ticket history and drafting customer responses.	Apply data minimization on retrieved records; verify customer identity and block cross-account data disclosure.
Research assistants	Searching public repositories and synthesizing external articles.	Isolate retrieved web content as untrusted data; block any tool calls triggered solely by external document text.

6.4 Future research directions

Several extensions warrant further investigation:

- **Typed Tool Schemas and Policy Templates:** Developing standardized, machine-verifiable tool metadata schemas alongside domain-specific policy templates for healthcare, finance, education, and enterprise administration.
- **Multi-Agent Delegation Security:** Extending action gating to multi-agent pipelines where an orchestrator delegates subtasks to specialized sub-agents, ensuring authority strictly attenuates across delegation hops rather than escalating.
- **Multimodal Injection Resilience:** Broadening boundary enforcement to multimodal agents where malicious instructions may be embedded in images, audio streams, or structured attachments, enforcing authorization independently of input modality.

7 Conclusion

Indirect prompt injection poses a fundamental security challenge for tool-using AI agents because natural-language interfaces blur the boundary between trusted user instructions and untrusted external data. When an agent possesses access to private records and external APIs, instruction confusion can escalate into unauthorized data disclosure, unsolicited communication, or destructive state modification.

Review of existing literature and benchmarks—including InjecAgent, AgentDojo, Spotlighting, Task Shield, and industry least-privilege guidelines—demonstrates that while prompt-level structuring and content detection reduce susceptibility, they cannot serve as standalone security guarantees against adaptive attacks. Conversely, rigid static permissions or indiscriminate confirmation prompts impair legitimate utility.

To reconcile security and usability, this paper proposed a conceptual Risk-Aware Action Gating framework that interposes an independent policy enforcement layer between agent planning and tool execution. By evaluating user intent, session scope, action sensitivity, target destination, and security policy, the gate classifies every proposed tool call into ALLOW, REQUIRE APPROVAL, or BLOCK, while ensuring external data can never grant new authority. Although empirical implementation and benchmark validation remain future work, this architecture establishes a structured foundation for building trustworthy, auditable, and resilient tool-using AI agents.

References

[1] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've *signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection*," in Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 2023, pp. 79–90, doi: 10.1145/3605764.3623985.

[2] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *Proceedings of the* 33rd USENIX Security Symposium, 2024, pp. 1831–1847.

[3] L. Beurer-Kellner, B. Buesser, A. M. Creţu, E. Debenedetti, D. Dobos, D. Fabian, M. Fischer, D. Froelicher, K. Grosse, D. Naeff, E. Ozoani, A. Paverd, F. Tramèr, and V. Volhejn, "Design patterns for securing LLM agents against prompt injections," *arXiv preprint arXiv:2506.08837*, 2025. [Online]. Available: https://arxiv.org/abs/2506.08837

[4] OWASP Foundation, "LLM Prompt Injection Prevention Cheat Sheet," *OWASP Cheat Sheet Series*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/LLM_Prompt_Injection_Prevention_Cheat_Sheet.html

[5] R. Bhagwatkar, K. Kasa, A. Puri, G. Huang, I. Rish, G. W. Taylor, K. D. Dvijotham, and A. Lacoste, "Indirect prompt injections: Are firewalls all you need, or stronger benchmarks?" *arXiv preprint arXiv:2510.05244*, 2025. [Online]. Available: https://arxiv.org/abs/2510.05244

[6] J. Yi, Y. Xie, B. Zhu, E. Kiciman, G. Sun, X. Xie, and F. Wu, "Benchmarking and defending against indirect prompt injection attacks on large language models," in *Proceedings of the* 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2025, pp. 1809–1820, doi: 10.1145/3690624.3709179.

[7] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents," in *Findings of the Association for Computational Linguistics: ACL* 2024, 2024, pp. 10471–10506, doi: 10.18653/v1/2024.findingsacl.624.

[8] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," in *Advances in Neural Information Processing Systems*, vol. 37, 2024, doi: 10.52202/0790172636.

[9] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending against indirect prompt injection attacks with spotlighting," *arXiv preprint arXiv:2403.14720*, 2024. [Online]. Available: https://arxiv.org/abs/2403.14720

[10] F. Jia, T. Wu, X. Qin, and A. Squicciarini, "The Task Shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents," in *Proceedings of the* 63rd Annual Meeting of the Association for Computational Linguistics, 2025, pp. 29680–29697, doi: 10.18653/v1/2025.acllong.1435.