US JOURNAL OF NEW INSIGHTS IN TECH & EDUCATION

Vol. 6, No. 1 · Article ID: USJNITE0601 · Journal Homepage: www.edtechinsights.org

Prompt Injection Attacks and Defense Mechanisms in LLM-Based AI Agents: A Systematic Review

Kanita Haider[1], Oishe Al Mariz[2]*, Mahfuzulhoq Chowdhury[3]

[123]Department of Computer Science & Engineering, Faculty of Electrical & Computer Eng., Chittagong University of Engineering & Technology, Chattogram, Bangladesh

** oishe4242@gmail.com*

Received: 23 July 2026; Revised: 3 August 2026; Accepted: 6 August 2026; Published: 10 August 2026

KEYWORDS

Prompt Injection;

Large Language Models;

AI Agents;

LLM Security; Indirect Prompt Injection;

Adversarial Robustness

Abstract

Large language model (LLM) based AI agents are increasingly deployed to autonomously browse the web, call external tools, retrieve documents, and act on behalf of users, yet this same autonomy exposes them to prompt injection, a vulnerability class in which adversarial instructions embedded in user input or in externally retrieved content override the developer's original instructions and hijack the agent's behaviour. This paper presents a systematic review of the prompt injection literature published between 2022 and 2026, synthesising findings from more than thirty peer-reviewed and preprint studies to characterise the attack surface of LLM agents and the defenses proposed against it. Attacks are organised along two axes, direct versus indirect injection and manual versus optimization-based construction, while defenses are grouped into four families: prompt-level heuristics, training-time alignment, detection and filtering, and architectural isolation. Drawing on benchmark results reported across the reviewed studies, the review shows that attack success rates remain high across agent architectures, ranging from roughly twenty to over ninety percent depending on the tool environment and adversary capability, and that adaptive attackers routinely defeat single-layer defenses. Architectural approaches that separate control flow from untrusted data, such as capability-based sandboxing and dual-LLM designs, offer the strongest guarantees but at a cost to agent utility and engineering complexity. The review identifies a persistent gap between attack sophistication and defense maturity and recommends defense-in-depth strategies, standardized cross-benchmark evaluation, and human oversight of consequential agent actions as priorities for future research and deployment.

1. Introduction

Large language models (LLMs) have moved rapidly from single-turn conversational tools into the reasoning core of autonomous software agents. Modern LLM-based agents plan multi-step tasks, call external application programming interfaces (APIs), browse the open web, read email and calendar data, execute code, and increasingly coordinate with other agents through emerging protocols such as the Model Context Protocol. This shift from passive text generation to autonomous action fundamentally changes the risk profile of the underlying model: an LLM that merely answers a question can, at worst, produce an incorrect or biased response, but an LLM that can send an email, transfer funds, or modify a file can cause direct, real-world harm if it is manipulated into acting against its user's interests. The vulnerability that most directly threatens this new class of system is prompt injection, first named and described by Willison (2022) and formally analysed by Perez and Ribeiro (2022), who demonstrated that carefully crafted natural-language instructions could override a model's original system prompt with striking reliability.

At the heart of prompt injection lies an architectural weakness that is easy to state but hard to fix: contemporary LLMs receive developer instructions, user input, and any retrieved or tool-returned content as a single, undifferentiated stream of tokens. Because the model has no reliable mechanism for distinguishing an instruction it should obey from data it should merely process, an attacker who can influence any part of that stream, whether by typing directly into a chat box or by planting text inside a web page, document, email, or database record that the agent will later read, can potentially redirect the model's behaviour. Greshake et al. (2023) were the first to demonstrate this indirect variant at scale, showing that real-world LLM-integrated applications could be remotely compromised through content the attacker never needed to type into the interface at all. Since that publication, indirect prompt injection has become the dominant threat model for agentic systems, because agents are, by design, constantly exposed to untrusted external content.

The practical consequences of this vulnerability are no longer theoretical. Researchers have shown that indirect prompt injection can be used to exfiltrate personal data observed by tool-calling banking agents (Alizadeh et al., 2025), to manipulate automated peer review of scientific manuscripts by hiding instructions in a paper's text so effectively that acceptance scores rose from an average of 5.34 to 7.99 out of ten (Keuper, 2025), and to compromise navigation and mobile-operating-system agents with attack success rates exceeding eighty-five percent in controlled evaluations. Industry incident reports catalogued in recent comprehensive reviews document remote code execution vulnerabilities in widely used coding assistants and unintended disclosure of licensing information

by consumer chatbots (Gulyamov et al., 2026), underscoring that the threat spans consumer, enterprise, and developer-tooling contexts alike. The Open Worldwide Application Security Project has accordingly ranked prompt injection as the foremost security risk facing LLM-integrated applications, a designation that has driven a rapid expansion of both attack research and defensive engineering since 2023.

Despite this attention, the defense literature remains fragmented. Proposals range from simple prompt-engineering heuristics, such as wrapping untrusted data in delimiters or appending reminder instructions, to substantial retraining of the underlying model, to entirely new agent architectures that attempt to prevent untrusted data from ever influencing control flow. Each family of defenses reports different threat models, different benchmarks, and different metrics, which makes it difficult for a practitioner deciding how to secure a production agent, or a researcher deciding where to focus new work, to compare approaches on equal footing. This fragmentation motivates the present review.

This paper makes three contributions. First, it synthesises the attack literature into a coherent taxonomy that distinguishes attacks by injection vector (direct versus indirect), construction method (manual, template-based, or optimization-based), and target (single-agent versus multi-agent systems). Second, it organises the corresponding defense literature into four mechanistic families and compares their reported effectiveness, utility cost, and generalisation properties using evidence drawn from more than thirty studies published between 2022 and 2026. Third, it identifies the concrete gaps that remain between what current defenses can guarantee and what production agent deployments actually require, and translates those gaps into research and deployment recommendations. Section 2 reviews the relevant literature. Section 3 describes the review methodology. Section 4 presents a comparative analysis supported by a taxonomy table and five figures. Section 5 concludes with recommendations.

2. Literature Review

The literature on prompt injection can be organised chronologically and thematically into five overlapping strands: foundational characterisations of the vulnerability, taxonomies and formal frameworks, benchmark and evaluation platforms, defense mechanisms, and case studies documenting real-world exploitation. Figure 1 situates the studies reviewed in this paper along a timeline, illustrating how the field moved from informally naming the vulnerability in 2022 to producing benchmarked defenses and architectural solutions by 2025 and 2026.

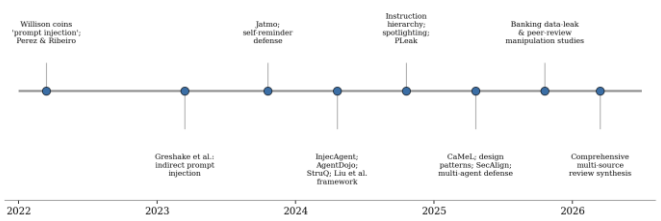

Figure 1: Timeline of key milestones in prompt injection attack and defense research, 2022–2026, based on the studies reviewed in this paper.

2.1 Foundational Work and Threat Characterisation

Perez and Ribeiro (2022) provided the first systematic study of what they termed 'goal hijacking' and 'prompt leaking', demonstrating that short adversarial instructions appended to otherwise benign prompts could reliably override a deployed model's original task. Around the same time, Willison (2022) coined the term 'prompt injection' itself, drawing an explicit analogy to SQL injection: just as a web application that concatenates untrusted user input directly into a SQL query can be hijacked by an attacker who supplies syntactically meaningful input, an LLM application that concatenates untrusted text into its prompt can be hijacked by an attacker who supplies semantically meaningful instructions. This analogy proved durable and now frames nearly all subsequent discussion of the vulnerability. Greshake et al. (2023) extended the threat model from direct injection, where the attacker is the user typing into the interface, to indirect injection, where the attacker instead poisons a data source that the LLM application will later retrieve and process without the user's direct involvement. Their taxonomy of resulting harms, including data theft, remote control of the agent, and self-propagating 'worm' behaviour across interconnected agents, remains the reference point for nearly every later paper in the field, including the multi-agent propagation study of Lee and Tiwari (2024), who showed empirically that a single compromised agent could silently spread malicious instructions across a network of otherwise isolated agents by exploiting inter-agent communication channels.

Adjacent to prompt injection sits the older literature on adversarial examples in machine learning generally. Goodfellow, Shlens, and Szegedy (2015) and Madry, Makelov, Schmidt, Tsipras, and Vladu (2018) established the foundational vocabulary of adversarial perturbation and adversarial training that later optimization-based prompt injection and jailbreak research draws upon directly. Zou, Wang, Carlini, Nasr, Kolter, and Fredrikson (2023) applied this optimization mindset to language models with the Greedy Coordinate Gradient method, producing universal adversarial suffixes that transfer across models and, while originally framed as a jailbreak technique, established the methodological template that later optimization-based prompt injection attacks (e.g., Shi et al., 2024) would adopt.

2.2 Taxonomies, Surveys, and Formal Frameworks

As the literature matured, several groups produced formal frameworks intended to unify disparate attacks under a common notation. Liu, Jia, Geng, Jia, and Gong (2024) proposed a general formalisation in which existing attacks, including naive concatenation, escape-character, context-ignoring, and fake-completion attacks, are expressed as special cases of a single parameterised construction; using this framework they conducted a systematic evaluation of five attacks and ten defenses across ten LLMs and seven tasks, concluding that none of the evaluated defenses reliably generalised across attacks. Zverev, Abdelnabi, Fritz, and Lampert (2024) probed a more fundamental question, whether current LLMs are even capable, in principle, of separating instructions from data when explicitly trained or prompted to do so, and found that separation remains imperfect even under favourable conditions, a result that helps explain why purely prompt-level defenses struggle. Broader surveys, including Xu and Parhi (2025) on the full lifecycle of LLM attacks and Gulyamov et al. (2026), who synthesised over one hundred sources spanning academic papers, industry disclosures, and documented incidents such as a critical remote-code-execution vulnerability in a popular coding assistant, situate prompt injection within the wider landscape of LLM and agent security and reinforce that the vulnerability spans the entire deployment stack rather than a single component.

2.3 Benchmarks and Empirical Evaluation Platforms

A recurring theme in the literature is the difficulty of measuring attack and defense effectiveness consistently. Zhan, Liang, Ying, and Kang (2024) addressed this with InjecAgent, a benchmark of 1,054 test cases spanning seventeen user-facing tools and sixty-two attacker tools, categorising attack goals into direct user harm and private-data exfiltration; their evaluation of thirty LLM agents found that a ReAct-prompted GPT-4 agent was vulnerable roughly twenty-four percent of the time, a figure that nearly doubled when the injected instruction was reinforced with an explicit 'hacking prompt'. Debenedetti et al. (2024) introduced AgentDojo, a dynamic environment simulating realistic agentic tasks such as banking, travel booking, and workspace management, allowing both attacks and defenses to be evaluated against genuine multi-step tool-use trajectories rather than single-turn prompts; AgentDojo has since become a common yardstick against which architectural defenses such as CaMeL (Debenedetti, Shumailov et al., 2025) are validated. A follow-up study extended InjecAgent's threat model to externally stored personal data and required actual data leakage, rather than mere task hijacking, to count as a successful attack, evaluating six LLMs and four defense strategies across an expanded forty-eight-task banking suite. Complementing these general-purpose benchmarks, narrower evaluations have probed specific agent classes: a navigation-agent attack (PINA) achieved an average attack success rate of 87.5 percent against embodied agents operating under black-box, long-context constraints, while environment-injection attacks against mobile-operating-system agents have been reported to succeed in up to ninety-three percent of trials.

2.4 Defense Mechanisms

Proposed defenses fall broadly into four mechanistic families, summarised visually in Figure 5 later in this paper. Prompt-level heuristic defenses modify the prompt construction process without retraining the model: examples include enclosing untrusted data in explicit delimiters, appending an instruction reminding the model to ignore embedded commands (the 'self-reminder' defense of Xie et al., 2023, shown to reduce jailbreak success in ChatGPT), and spotlighting, in which Hines, Lopez, Hall, Zarfati, Zunger, and Kiciman (2024) transform untrusted input so the model can visually and statistically distinguish it from trusted instructions. These approaches are inexpensive to deploy but, as Liu et al. (2024) and subsequent adaptive-attack studies show, are readily bypassed by an attacker aware of the specific heuristic in use.

Training-time defenses instead modify the model's parameters so that it learns to privilege developer instructions over embedded ones. Chen, Piet, Sitawarin, and Wagner (2025) introduced StruQ, which separates prompts and data into distinct channels at the front end and fine-tunes the model on a structured-instruction-tuning dataset containing injected instructions in the data channel that the model is trained to ignore, reducing attack success rates to under two percent against optimization-free attacks with minimal utility loss. Piet et al. (2023) proposed Jatmo, which sidesteps the general instruction-following capability altogether by fine-tuning task-specific models that never learn to follow arbitrary instructions embedded in their input. Wu et al. (2024) introduced instructional segment embeddings that explicitly encode the provenance (system, user, or data) of each token, while the closely related SecAlign defense of Chen et al. (2025) applies preference optimization to directly penalise the model for following injected instructions. Yi et al. (2023) benchmarked a range of both training-time and prompt-level defenses specifically against indirect injection and found that combinations of techniques outperformed any single approach, foreshadowing the defense-in-depth conclusion of later work.

Detection and filtering defenses attempt to identify malicious content before or after it reaches the model, ranging from lightweight classifiers such as Meta AI's (2024) PromptGuard family to more elaborate multi-agent detection pipelines. Hossain et al. (2025) proposed a multi-agent defense pipeline in which specialised detector agents, arranged in either a sequential chain or a hierarchical coordinator topology, screen incoming content in real time before it reaches the primary task-performing agent. Detection-based defenses generalise better across attack styles than static heuristics but introduce additional latency and remain vulnerable to adversarially optimized inputs specifically designed to evade the detector, an arms-race dynamic documented in the optimization-based attack literature (Shi et al., 2024; Hui et al., 2024).

The most structurally ambitious family of defenses abandons the attempt to make the LLM itself robust and instead redesigns the surrounding agent architecture so that untrusted data can never influence consequential actions, regardless of what the model is told. Beurer-Kellner et al. (2025) formalised six such design patterns, including preventing any feedback from tool outputs back into the planning model and enforcing a plan-then-execute discipline. The most fully developed instance of this approach, CaMeL (Debenedetti, Shumailov, et al., 2025), separates a privileged planning model with no exposure to untrusted tokens from a quarantined model that processes untrusted content but has no tool access, enforcing information-flow-control policies through a custom interpreter; the authors report that CaMeL 'practically solves' the security evaluation of the AgentDojo benchmark. This capability-based approach traces conceptually to Willison's (2023) earlier proposal of a 'dual LLM' pattern and represents the clearest example of trading engineering complexity and some agent flexibility for strong, verifiable security guarantees rather than probabilistic robustness.

2.5 Case Studies and Real-World Exploitation

A growing body of applied work demonstrates that these vulnerabilities are exploitable outside the laboratory. Keuper (2025) showed that hidden text embedded in a scientific manuscript's PDF could manipulate large-language-model-generated peer reviews with near-total reliability. Alizadeh, Samei, Stetsenko, and Gilardi (2025) demonstrated data-flow-based exfiltration attacks against a fictitious tool-calling banking agent built on the AgentDojo benchmark, showing that even simple injected instructions could cause an agent to leak personal data it had legitimately observed during task execution. Hui et al. (2024) documented prompt-leaking attacks capable of extracting proprietary system prompts from deployed LLM applications. Gulyamov et al. (2026) catalogue further documented incidents, including a critical remote-code-execution vulnerability in a widely deployed coding assistant and unintended disclosure of licensing data by a consumer chatbot, underscoring that indirect prompt injection has produced concrete, disclosed vulnerabilities in production systems.

2.6 Literature Gap

Three gaps recur across the reviewed literature. First, evaluation remains inconsistent: different studies use different benchmarks, threat models, and success criteria, which makes cross-study comparison of defense effectiveness unreliable. Second, nearly every probabilistic defense, whether prompt-level, training-time, or detection-based, has subsequently been shown to degrade substantially against an adaptive attacker, with reported bypass rates exceeding fifty percent in some evaluations; only architectural, capability-based approaches such as CaMeL currently offer guarantees that do not degrade

under adaptive pressure, and these come with meaningfully higher engineering cost and reduced agent flexibility. Third, the literature has only recently begun to examine multi-agent and cross-context propagation (Lee & Tiwari, 2024; Gulyamov et al., 2026), leaving open how injected instructions spread through the increasingly common networks of interacting agents. This review's analysis in Section 4 is structured explicitly around these three gaps.

2.2 Experimental / Analytical Procedure

Describe the methodology, experimental setup, or analytical/statistical procedure used to obtain the results. This is demo placeholder text only.

3. Methodology

This paper follows a structured narrative literature review methodology appropriate for a fast-moving, cross-disciplinary security topic where formal meta-analysis of quantitative effect sizes is not yet feasible because reviewed studies use heterogeneous benchmarks, agent environments, and success metrics. The review process comprised four stages: search, screening, thematic coding, and comparative synthesis.

Search. Candidate studies were identified through iterative keyword searches conducted against arXiv, the ACL Anthology, USENIX Security proceedings, ACM Digital Library, and general web search covering September 2022 through mid-2026. Search terms included combinations of 'prompt injection', 'indirect prompt injection', 'LLM agent security', 'jailbreak defense', 'instruction hierarchy', and the names of specific benchmarks and defenses identified during initial reading and followed forward and backward through citation chains.

Screening. A study was included if it (a) proposed, formalised, benchmarked, or defended against an attack in which adversarial natural-language content alters an LLM's or LLM agent's intended behaviour without authorised modification of the model's system-level instructions, and (b) reported a reproducible method, empirical result, or design artefact. Preference was given to peer-reviewed venue publications where available, supplemented by arXiv preprints for the most recent 2025–2026 work, which in this fast-moving subfield frequently precedes formal publication by many months.

Thematic coding. Each included study was coded along four dimensions: injection vector (direct or indirect), construction method (manual, template-based, or optimization-based), defense family (prompt-level heuristic, training-time alignment, detection and filtering, or architectural isolation), and evaluation context (single-agent, tool-integrated agent, or multi-agent system). This scheme forms the basis of Table 1 and Figures 2 through 5 in Section 4.

Comparative synthesis. Because reviewed studies rarely share a common benchmark, this review does not attempt a quantitative meta-analysis of attack success rates in the statistical sense. Instead, Section 4 reports figures as originally stated by each study's authors, explicitly attributed to the study and its evaluation context, and synthesises qualitative patterns that recur despite methodological heterogeneity. Limitations of this approach, principally the risk of selection bias and the incomparability of headline percentages across differing threat models, are acknowledged alongside the review's conclusions in Section 5.

4. Analysis / Results Analysis

This section synthesises the coded evidence from the reviewed studies into a taxonomy of attack vectors, a comparative assessment of defense families, and a discussion of the utility-security trade-off that runs through the literature.

4.1 Taxonomy of Attack Vectors

Table 1 summarises the attack taxonomy that emerged from thematic coding of the reviewed literature, and Figure 2 illustrates schematically how direct and indirect pathways both converge on the same underlying weakness: an LLM agent that cannot structurally distinguish trusted instructions from untrusted data.

Attack Category	Subtype / Mechanism	Description	Representative Study
Direct Injection	Goal hijacking / instruction override	Attacker types adversarial instructions straight into the user-facing prompt to override the system prompt.	*Perez & Ribeiro (2022)*
Direct Injection	Prompt leaking	Attacker extracts the confidential system prompt or proprietary instructions from the application.	*Hui et al. (2024)*
Indirect Injection	Retrieved-content injection	Malicious instructions are planted in a web page, document, or search result later retrieved by the agent.	*Greshake et al. (2023)*
Indirect Injection	Tool-output injection	Malicious instructions are embedded in the output of a tool call (e.g., email body, file contents).	*Zhan et al. (2024)*
Indirect Injection	Data-flow exfiltration	Injection is engineered specifically to cause leakage of private data the agent has legitimately accessed.	*Alizadeh et al. (2025)*

Attack Category	Subtype / Mechanism	Description	Representative Study
Optimization-Based	Gradient / suffix optimization	Adversarial token sequences are optimized against model gradients or outputs to maximise attack success.	*Zou et al. (2023); Shi et al. (2024)*
Multi-Agent Propagation	Self-replicating injection	A compromised agent forwards malicious instructions to other agents, propagating the compromise virally.	*Lee & Tiwari (2024)*
Domain-Specific	Embodied / navigation agent injection	Injected instructions redirect physically or virtually embodied agents (e.g., navigation, mobile OS control).	*PINA study (agent navigation, 2026)*

Table 1: Taxonomy of Prompt Injection Attack Vectors and Representative Studies

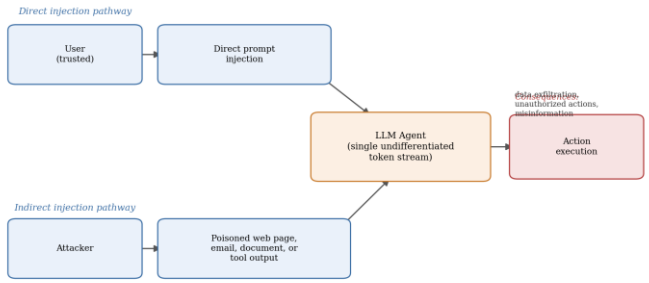

Figure 2: Schematic comparison of direct and indirect prompt injection pathways into an LLM agent, and the resulting downstream consequences once untrusted tokens influence tool or action execution.

Two structural distinctions dominate the taxonomy in Table 1 and Figure 2. The first, direct versus indirect injection, determines who the attacker needs to be: a direct attack requires the attacker to be the person typing into the interface, whereas an indirect attack, first formalised by Greshake et al. (2023), requires only that the attacker can influence some piece of content the agent will later ingest, a web page, a shared document, an email, a calendar invitation, or the output of a tool call. Indirect injection is the more consequential threat for agentic systems specifically because agents are architecturally designed to continuously ingest external content as part of normal operation, meaning the attack surface scales with every new data source or tool the agent is granted access to. The second distinction, manual versus optimization-based construction, determines how sophisticated the injected payload itself is. Manual and template-based attacks, of the kind catalogued by Perez and Ribeiro (2022) and formalised by Liu et al. (2024), rely on natural-language phrasing such as instruction-override statements or fake task-completion markers, and remain highly effective against undefended systems precisely because they require no special access to the target model. Optimization-based attacks, by contrast, use gradient information or black-box optimization loops, in the tradition established for adversarial examples generally by Goodfellow, Shlens, and Szegedy (2015) and Madry, Makelov, Schmidt, Tsipras, and Vladu (2018) and adapted to language models by Zou et al. (2023), to search for token sequences that reliably defeat a specific target model or a specific defense.

As Table 1 shows, a third and increasingly important category, multi-agent propagation, sits somewhat orthogonally to the vector/construction distinctions. Lee and Tiwari's (2024) Prompt Infection attack demonstrated that once a single agent in an interconnected multi-agent system is compromised via ordinary indirect injection, the compromise can self-replicate across agents that never directly ingested the original malicious content, behaving analogously to a computer virus propagating across a network. This finding is particularly significant given the accelerating industry adoption of multi-agent orchestration frameworks and standardised inter-agent communication protocols, which multiply the number of trust boundaries an attacker can exploit relative to a single-agent deployment.

4.2 Reported Attack Effectiveness

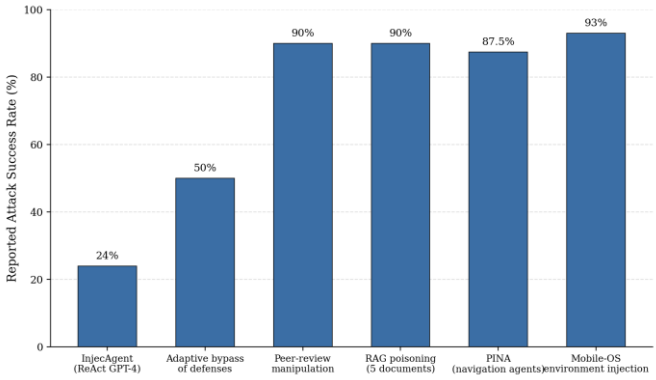

Figure 3: Reported attack success rates (ASR) across six representative studies in the reviewed literature. Percentages are as originally reported by each study's authors under their respective threat models and are not directly comparable across studies.

InjecAgent's evaluation of a ReAct-prompted GPT-4 agent under a realistic seventeen-tool environment found a comparatively modest twenty-four percent success rate for baseline indirect injection (Zhan et al., 2024), while the same study found that reinforcing the injected instruction with an explicit hacking-style prompt nearly doubled that figure. At the other extreme, retrieval-augmented generation systems were shown to be manipulable ninety percent of the time using as few as five adversarially crafted documents (Gulyamov et al., 2026), automated peer-review systems were manipulated with comparable reliability by hidden instructions embedded in manuscript text (Keuper, 2025), embodied navigation agents

were compromised 87.5 percent of the time by the PINA attack, and environment-injection attacks against mobile-operating-system agents reached ninety-three percent success. Critically, adaptive attacks that are specifically constructed with knowledge of a target defense have been reported to bypass that defense more than fifty percent of the time even when the defense performs well against non-adaptive attacks (a pattern reported consistently across the surveyed literature, e.g., Liu et al., 2024).

Two patterns are worth drawing out from these figures. First, success rates rise sharply once the attacker is permitted to reinforce the injected instruction or to optimize it against the specific target system, suggesting that headline success rates measured against naive, non-adaptive attacks systematically understate the risk faced by a deployed system against a motivated adversary. Second, success rates vary considerably by domain: benchmarks that measure narrow, well-defined outcomes such as a review score or a navigation waypoint tend to report higher success rates than benchmarks such as InjecAgent that measure completion of an entire multi-step harmful task, because the latter requires the injected instruction to survive the agent's full planning and tool-execution loop rather than influencing a single output.

4.3 Comparative Assessment of Defense Families

The reviewed defense literature clusters into four mechanistic families that trade off differently along the axes of deployment cost, effectiveness against adaptive attackers, and impact on agent utility. Figure 4 presents a qualitative, author-synthesised comparison of the four families across five dimensions drawn from the patterns reported across the reviewed studies; the ratings are illustrative rather than measured on a common empirical scale, since no single study in the corpus benchmarks all four families under identical conditions.

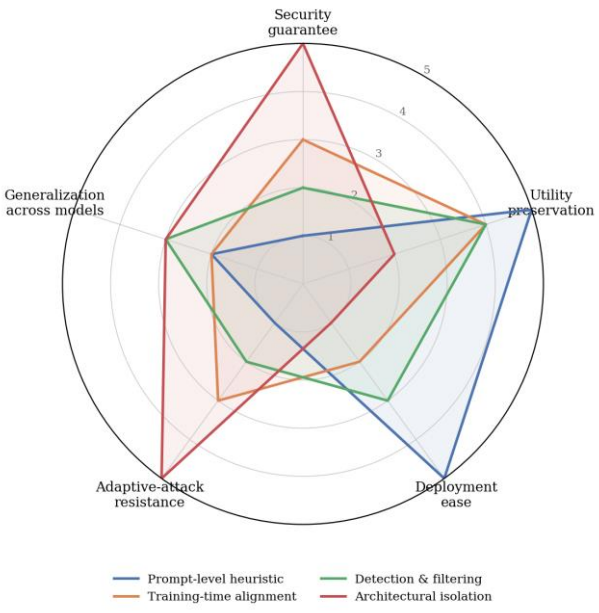

Figure 4: Qualitative comparison of the four defense families across five dimensions, synthesised by the authors from the patterns reported across the reviewed literature (illustrative 1–5 scale, not a measured empirical benchmark).

Prompt-level heuristic defenses, including delimiter-based data marking, self-reminder instructions (Xie et al., 2023), and spotlighting transformations (Hines et al., 2024), require no model retraining and can be deployed immediately in front of any existing model, which makes them attractive for rapid mitigation. However, the reviewed evidence consistently shows that these defenses are heuristic rather than guaranteed: because they operate by nudging the same underlying token stream that the attacker is also manipulating, a sufficiently informed adversary can construct payloads that survive the specific transformation in use, and Liu et al.'s (2024) systematic evaluation of ten such defenses found none that generalised reliably across the five attack types they tested.

Training-time defenses achieve stronger and more consistent results by changing what the model has learned to do rather than merely how the prompt is formatted. StruQ's structured instruction tuning (Chen, Piet, Sitawarin, & Wagner, 2025) reduced attack success rates to under two percent against optimization-free attacks while preserving general-purpose utility, and related approaches such as instructional segment embeddings (Wu et al., 2024) and preference-optimization-based alignment (Chen et al., 2025, SecAlign) push in the same direction. Task-specific fine-tuning approaches such as Jatmo (Piet et al., 2023) go further still by eliminating general instruction-following capability from the deployed model altogether for narrow applications, at the cost of losing the flexibility that makes general-purpose LLM agents useful in the first place. The principal limitation of this family is that training-time defenses are specific to the model they were applied to; each new base model or fine-tuning run must be re-secured, and optimization-based adaptive attacks explicitly designed to evade a trained defense continue to erode the guarantees these methods offer over time.

Detection and filtering defenses, ranging from lightweight open classifiers such as Meta AI's (2024) PromptGuard to the coordinated multi-agent detection pipelines proposed by Hossain et al. (2025), sit in between the previous two families: they can be updated independently of the primary model and can, in principle, generalise across models, but they introduce additional inference latency and shift the security problem rather than eliminating it, since the detector itself becomes a target for adversarial evasion. Studies of optimization-based attacks against LLM-as-a-judge systems (Shi et al., 2024) and prompt-leaking attacks (Hui et al., 2024) illustrate that classifiers and judges built from LLMs inherit the same instruction/data-conflation weakness they are meant to guard against.

Architectural isolation defenses represent the most significant departure from the preceding three families because they do not attempt to make the model robust at all; instead, they redesign the surrounding system so that no amount of successful manipulation of the model's output can translate into an unauthorised action. The design patterns catalogued by Beurer-Kellner et al. (2025) and their fullest realisation in CaMeL (Debenedetti, Shumailov, et al., 2025) are reported to 'practically solve' the AgentDojo benchmark's security evaluation, a claim substantially stronger than anything reported for the other three defense families. This strength comes at a real cost: architectural defenses require the developer to formally specify the space of permitted actions and data flows in advance, which constrains the open-ended flexibility that makes general-purpose agents attractive, and existing implementations remain comparatively early-stage relative to the prompt-level and training-time defenses already deployed in commercial products.

4.4 The Utility-Security Trade-off

Across every defense family reviewed, the same underlying trade-off recurs, illustrated as a layered defense-in-depth model in Figure 5: measures that provide stronger security guarantees tend to reduce the flexibility, generality, or responsiveness that makes LLM agents useful, while measures that preserve full agent flexibility tend to offer only probabilistic, attacker-

dependent protection. No single defense family reviewed in this literature dominates on all dimensions simultaneously, which is the central empirical basis for the defense-in-depth recommendation developed in Section 5.

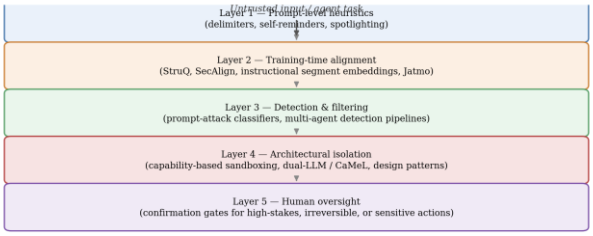

Figure 5: A defense-in-depth model synthesising the four mechanistic defense families reviewed in this paper, together with human oversight as a final layer for high-stakes or irreversible agent actions.

5. Conclusion and Recommendations

Prompt injection is not a peripheral bug in large language model applications but a direct consequence of the architecture that makes LLMs useful: their inability, absent specific countermeasures, to distinguish trusted instructions from untrusted data within a single stream of tokens. This review has synthesised more than thirty studies published between 2022 and 2026 to show that this weakness has been exploited across an increasingly wide range of domains, from academic peer review and banking agents to embodied navigation and mobile-operating-system control, with reported attack success rates ranging from roughly twenty percent in realistic, multi-step agentic environments to over ninety percent in narrower, single-outcome settings, and that adaptive attackers routinely defeat defenses that perform well against naive attacks alone.

The comparative analysis in Section 4 supports several concrete recommendations for practitioners building or deploying LLM-based agents. First, no single defense family reviewed here should be treated as sufficient in isolation; the strongest evidence in the literature supports layered, defense-in-depth deployments (Figure 5) that combine training-time alignment, detection and filtering, and, for any agent with access to consequential actions, architectural isolation patterns that constrain what an agent can do regardless of what it is told. Second, human confirmation should remain a mandatory gate for any agent action that is high-stakes, irreversible, or involves the transfer of sensitive data, since no current automated defense reliably prevents a sufficiently motivated and informed adversary from manipulating agent behaviour. Third, organisations deploying agents that interact with multiple other agents or automated systems should specifically evaluate resilience to propagation attacks of the kind demonstrated by Lee and Tiwari (2024), since this threat class has received comparatively little defensive attention relative to single-agent injection.

For the research community, this review points to three priorities. Standardisation of evaluation is the most pressing: the field would benefit substantially from wider adoption of shared, realistic benchmarks such as AgentDojo and InjecAgent, reported alongside adaptive-attack results rather than naive-attack results alone. Architectural defenses deserve continued investment precisely because they are the only family reviewed here that offers guarantees rather than probabilities, and closing the gap between their strong security properties and their current cost to agent flexibility would meaningfully change the risk calculus for production deployment. Finally, multi-agent and cross-protocol propagation remains comparatively underexplored and merits urgent attention given the pace at which multi-agent orchestration and standardised inter-agent communication protocols are being adopted in industry.

This review has limitations that should temper its conclusions. Because the reviewed studies use heterogeneous benchmarks, threat models, and success metrics, the attack success rates compared in Section 4 are illustrative of overall patterns rather than directly comparable statistics, and a formal quantitative meta-analysis was not attempted for this reason. The review is also necessarily a snapshot of a fast-moving field in which new attacks and defenses continue to appear at a rapid pace; several of the studies cited here were preprints at the time of writing and may be revised prior to formal publication. Notwithstanding these limitations, the consistency of the central finding, that architectural, capability-based defenses currently offer the strongest and most durable guarantees while prompt-level and even many training-time defenses remain vulnerable to adaptive attackers, across studies with otherwise very different methodologies gives reasonable confidence that this pattern reflects a genuine and persistent property of the underlying problem rather than an artefact of any single benchmark or research group.

References

[1] Alizadeh, M., Samei, Z., Stetsenko, D., & Gilardi, F. (2025). Simple prompt injection attacks can leak personal data observed by LLM agents during task execution. arXiv preprint arXiv:2506.01055.

[2] Beurer-Kellner, L., Buesser, B., Creţu, A.-M., Debenedetti, E., Dobos, D., Fabian, D., Fischer, M., Froelicher, D., Grosse, K., Naeff, D., Ozoani, E., Paverd, A., Tramèr, F., & Volhejn, V. (2025). Design patterns for securing LLM agents against prompt injections. arXiv preprint arXiv:2506.08837.

[3] Chen, S., Piet, J., Sitawarin, C., & Wagner, D. (2025). StruQ: Defending against prompt injection with structured queries. Proceedings of the 34th USENIX Security Symposium.

[4] Chen, S., Zharmagambetov, A., Mahloujifar, S., Chaudhuri, K., Wagner, D., & Guo, C. (2025). SecAlign: Defending against prompt injection with preference optimization. Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, 2833–2847.

[5] Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., & Tramèr, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. Advances in Neural Information Processing Systems 37 (Datasets and Benchmarks Track).

[6] Debenedetti, E., Shumailov, I., Fan, T., Hayes, J., Tramèr, F., Balunović, M., & Kern, D. (2025). Defeating prompt injections by design. arXiv preprint arXiv:2503.18813.

[7] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. Proceedings of the International Conference on Learning Representations (ICLR).

[8] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 79–90.

[9] Gulyamov, S., Gulyamov, S., Rodionov, A., Khursanov, R., Mekhmonov, K., Babaev, D., & Rakhimjonov, A. (2026). Prompt injection attacks in large language models and AI agent systems: A comprehensive review of vulnerabilities, attack vectors, and defense mechanisms. Information, 17(1), 54.

[10] Hines, K., Lopez, G., Hall, M., Zarfati, F., Zunger, Y., & Kiciman, E. (2024). Defending against indirect prompt injection attacks with spotlighting. arXiv preprint arXiv:2403.14720.

[11] Hossain, S. M. A., Shayoni, R. K., Ameen, M. R., Islam, A., Mridha, M. F., & Shin, J. (2025). A multi-agent LLM defense pipeline against prompt injection attacks. arXiv preprint arXiv:2509.14285.

[12] Hui, B., et al. (2024). PLeak: Prompt leaking attacks against large language model applications. Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security.

[13] Keuper, J. (2025). Prompt injection attacks on LLM generated reviews of scientific publications. arXiv preprint arXiv:2509.10248.

[14] Lee, D., & Tiwari, M. (2024). Prompt infection: LLM-to-LLM prompt injection within multi-agent systems. arXiv preprint arXiv:2410.07283.

[15] Liu, Y., Jia, Y., Geng, R., Jia, J., & Gong, N. Z. (2024). Formalizing and benchmarking prompt injection attacks and defenses. Proceedings of the 33rd USENIX Security Symposium, 1831–1847.

[16] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. Proceedings of the International Conference on Learning Representations (ICLR).

[17] Meta AI. (2024). Llama Prompt Guard 2: A classifier model for detecting prompt attacks [Model card]. Hugging Face.

OWASP Foundation. (2025). OWASP Top 10 for Large Language Model Applications.

[18] Perez, F., & Ribeiro, I. (2022). Ignore previous prompt: Attack techniques for language models. arXiv preprint arXiv:2211.09527.

[19] Piet, J., Alrashed, M., Sitawarin, C., Chen, S., Wei, Z., Sun, E., Alomair, B., & Wagner, D. (2023). Jatmo: Prompt injection defense by task-specific finetuning. arXiv preprint arXiv:2312.17673.

[20] Shi, J., Yuan, Z., Liu, Y., Huang, Y., Zhou, P., Sun, L., & Gong, N. Z. (2024). Optimization-based prompt injection attack to LLM-as-a-Judge. Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security.

[21] Willison, S. (2022). Prompt injection attacks against GPT-3 [Blog post]. simonwillison.net.

[22] Willison, S. (2023). The dual LLM pattern for building AI assistants that can resist prompt injection [Blog post]. simonwillison.net.

[23] Wu, T., Zhang, S., Song, K., Xu, S., Zhao, S., Agrawal, R., Indurthi, S. R., Xiang, C., Mittal, P., & Zhou, W. (2024). Instructional segment embedding: Improving LLM safety with instruction hierarchy. arXiv preprint arXiv:2410.09102.

[24] Xie, Y., Yi, J., Shao, J., Curl, J., Lyu, L., Chen, Q., Xie, X., & Wu, F. (2023). Defending ChatGPT against jailbreak attack via self-reminders. Nature Machine Intelligence, 5(12), 1486–1496.

[25] Xu, W., & Parhi, K. K. (2025). A survey of attacks on large language models. arXiv preprint arXiv:2505.12567.

[26] Yi, J., Xie, Y., Zhu, B., Kiciman, E., Sun, G., Xie, X., & Wu, F. (2023). Benchmarking and defending against indirect prompt injection attacks on large language models. arXiv preprint arXiv:2312.14197.

[27] Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. Findings of the Association for Computational Linguistics: ACL 2024, 10471–10506.

[28] Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J. Z., & Fredrikson, M. (2023). Universal and transferable adversarial attacks on aligned language models. arXiv preprint arXiv:2307.15043.

[29] Zverev, E., Abdelnabi, S., Fritz, M., & Lampert, C. H. (2024). Can LLMs separate instructions from data? And what do we even mean by that? arXiv preprint arXiv:2403.06833.