BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Prompt Injection Attacks on AI Coding Assistants - *A Literature Survey and Layered Defense Framework*

[1]Anushka Kedar, [2]Prof. Ashima Mahendra

[1]Student (PRN: 72426615B), [2]Assistant Professor (Project Guide)
[1,2]Department of Artificial Intelligence and Data Science Engineering,
[1,2]Ajeenkya DY Patil School of Engineering, Pune, India
[1]anushkakedar2626@gmail.com, [2]ashimamahendra@dypic.in

Abstract

AI coding assistants are no longer limited to suggesting the next line of code. Many can inspect a repository, use external tools and execute commands for a developer. These features make them useful but also open up to a security concern: prompt injection. In this type of attack, untrusted text can steer a model away from the task it was given and may lead to actions such as exposing data or executing unsafe commands. This paper reviews eight studies published between 2022 and 2026, following the development of the topic from early chatbot attacks to attacks against agents that use tools. The studies are compared in a year-wise literature matrix and assessed across five capabilities. The review highlights four recurring gaps: limited evaluation focused specifically on coding assistants, insufficient testing against adaptive attackers, a separation between attack benchmarks and proposed defenses, and the absence of a combined defense architecture at the protocol level. Based on methods already discussed in the reviewed work, we outline a five-layer defense pipeline and suggest a way to evaluate it. This is a literature review not an implementation study; no model is trained and no new attack or defense is built nor tested.

Keywords: Prompt injection, AI coding assistants, large language models, Model Context Protocol, indirect prompt injection, agentic AI, literature survey, defense in depth

I. Introduction

AI-based coding tools such as GitHub Copilot, Cursor and Claude Code no longer just complete a line of code. They read whole projects, open a shell, and connect to external services through standard interfaces such as the Model Context Protocol (MCP) [1]. This is useful precisely because the model is left to decide, largely on its own, what to do next.

The fundamental problem is that a language model cannot reliably distinguish between instructions that come from a trusted source and the content it is asked to process [8], [5]. By embedding a malicious instruction in a file, issue, or tool response, an attacker can persuade the model to act on it as if it were part of the request. This is known as a prompt injection, and OWASP considers it a primary threat to applications built using large language models [2]. For an application that has access to the source code,

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

credentials, or command line of the coding assistant, the risks include leaking files, modifying dependencies, and executing unintended commands, possibly without the developer ever realizing what has happened.

This paper reviews eight studies that examine how prompt injection has been identified, tested, and addressed. The main contributions of the review are:

- A year-wise literature matrix that compares the methods, findings, and limitations of all eight studies, arranged from the newest (2026) to the oldest (2022) in Section III.
- A three-part taxonomy for coding-assistant attacks, based on how an attack is delivered, what form it takes, and whether its effects continue over time (Section II).
- A comparison of the studies and a discussion of four gaps that remain in the current research (Sections III–IV).
- A proposed five-layer defense pipeline, along with a suggested protocol for evaluating it (Sections V–VI).

The scope is deliberately limited to reviewing the literature and identifying research gaps. The proposed framework remains conceptual: it has not been implemented or tested. These limits are discussed in Section VII.

II. Background and Attack Taxonomy

A. From Autocomplete to Autonomous Agents

Coding assistants have evolved through multiple generations. The first ones could predict the next few tokens given context, while chat assistants let developers ask and answer questions in natural language and edit the code directly, allowing for injection attacks when trying to override the model's behavior. Agentic coding assistants take it a step further by being able to access and edit files within a project, run shell commands, and take advantage of tools via MCP with less human supervision [1]. Every additional feature grants the model more attack vectors and opportunities to have unintended consequences.

B. Why the Model Cannot Tell Code from Data

Applications built around large language models often combine the request of the developer with retrieved files and tool output in a single prompt. The model processes this as a sequence of tokens, without a reliable built-in signal that distinguishes between trusted statements and text that should be treated as data [5]. This is a significant difference to problems such as SQL injection, where parameterised queries make sure that commands and values are properly separated. In natural language there is no similarly dependable separation, and malicious instructions can be phrased in many different ways [1].

C. Attack Taxonomy

Group all prompt-injection attacks in Table I according to the three dimensions used in the reviewed papers: how the attack reaches the model, the form of the payload, and how long the attack lasts [1], [7]. Classifying attacks according to all three dimensions provides a more comprehensive overview than labeling each attack with a single category.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Table I: Three-dimensional Attack Taxonomy

Dimension	Categories	Representative example	Source
Delivery	Direct · Indirect · Protocol-level	Hidden instruction in a pull-request comment read by the agent	[7], [1]
Modality	Plain text · Encoded/obfuscated · Semantic	Base64-encoded payload decoded and followed by the model	[8], [4]
Propagation	Single-shot · Persistent · Viral	Payload written to agent memory, re-activating in a later session	[7], [1]

These dimensions may overlap; for instance, a malicious instruction embedded in the rules of a project can be both indirect (since it was not typed by the developer in the chat) and persistent (if it alters the configuration of the assistant for future interactions).

III. Literature Review

A. Method

This review encompasses the eight papers selected for the seminar, ranging from early controlled work on prompt injection to more recent work on agentic coding assistants. For each paper we have listed the approach taken, the main finding and stated or apparent limitation. Table II summarizes this information, with the studies ordered by date from most recent to earliest.

B. Survey Matrix

Table II: Year-wise Literature Survey Matrix

Author, year	Methodology and key findings	Limitations
Maloyan & Namiot, 2026 [1]	Systematisation of knowledge covering 78 sources; builds a three-axis delivery / modality / propagation taxonomy and a defense-in-depth outline, with new exploit chains for skill-based coding agents.	Secondary synthesis with no new experiments; some internal figures are inconsistent across the paper.
OWASP, 2025 [2]	Industry risk catalogue; ranks prompt injection as the top LLM application risk and lists seven mitigation practices, from input filtering to human approval.	Guidance document rather than an empirical study; gives no measured effectiveness for its own recommendations.
Zhan et al. (InjecAgent), 2024 [3]	Benchmark of 1,054 cases across 17 user-facing and 62 attacker-facing tools; measures how often 30 tool-calling agents can be hijacked through retrieved content.	Attacks limited to two steps within one interaction; only two fine-tuned agents were included.
Liu et al., 2024 [4]	Frames injection as an optimisation problem and uses a momentum-enhanced gradient search to craft a single injection string that transfers across tasks.	Needs white-box access to one small open-source model; not evaluated on commercial coding assistants.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Author, year	Methodology and key findings	Limitations
Hines et al., 2024 [5]	Proposes spotlighting — delimiting, datamarking and encoding — to mark untrusted text, cutting attack success sharply on GPT-family models.	Tested only against fixed, non-adaptive payloads; encoding reduces task accuracy on weaker models.
Sahoo et al., 2024 [6]	Broad survey of over forty prompting techniques, including retrieval-augmented generation and ReAct-style tool use, that explain how agentic behaviour arises.	Not a security study; included here only as background on why models gained tool-calling agency.
Greshake et al., 2023 [7]	First systematic demonstration that content retrieved from the web, email or code repositories can hijack LLM-integrated applications without the user typing anything malicious.	Mostly qualitative case studies; no statistical attack-success rates are reported.
Perez & Ribeiro, 2022 [8]	Earliest controlled study of direct prompt injection; defines goal hijacking and prompt leaking and tests both with a modular attack framework on GPT-3 models.	Hand-crafted attacks on an early, now largely superseded model family.

C. Capability Comparison

Table III provides a compact comparison of the same studies. “Yes” indicates that a capability is a main part of the paper, “Part” means it is discussed but is not a central focus, and “No” means the capability is not covered.

Table III: Capability Comparison of Reviewed Studies

Study	Ind.	Agt.	Auto.	Def.	Code
Maloyan & Namiot 2026 [1]	Yes	Yes	Part	Yes	Yes
OWASP 2025 [2]	Yes	Part	No	Yes	No
Zhan et al. 2024 [3]	Yes	Yes	No	No	No
Liu et al. 2024 [4]	Part	No	Yes	No	No
Hines et al. 2024 [5]	Yes	No	No	Yes	No
Sahoo et al. 2024 [6]	No	Part	No	No	No
Greshake et al. 2023 [7]	Yes	Part	No	No	Part
Perez & Ribeiro 2022 [8]	No	No	No	No	No
Proposed framework	Yes	Yes	Yes	Yes	Yes

Ind. means indirect delivery; Agt. means evaluation of agents or tool use; Auto. refers to automated or adaptive attacks; Def. indicates a proposed defense; and Code marks work focused specifically on coding assistants. The proposed framework row describes a design goal, not a result that has already been demonstrated.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

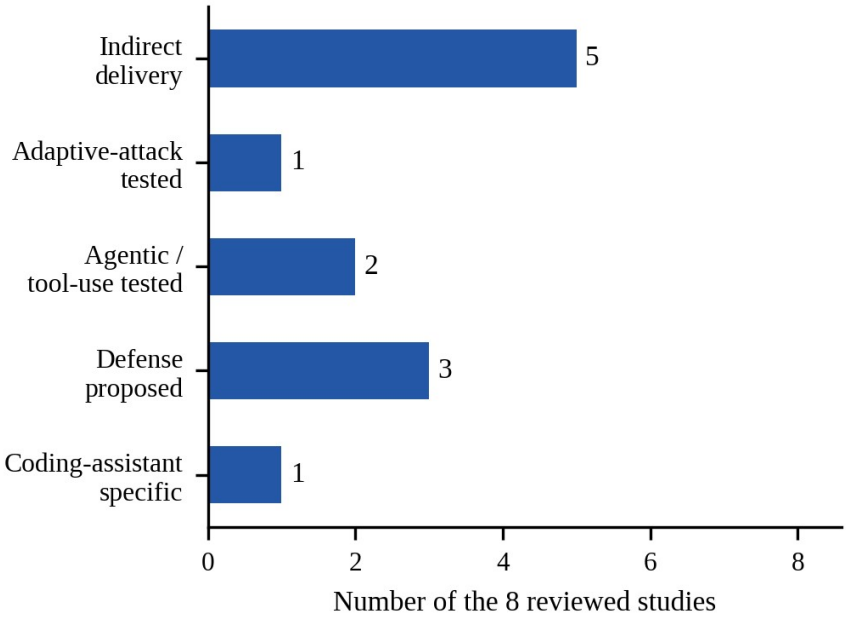


Fig. 1. Number of the eight reviewed studies that fully address each capability, derived from Table III.

D. Synthesis

Foundational attacks. Perez and Ribeiro [8] demonstrated, using examples, that instructions inserted directly in the prompt can steer a model toward unintended outputs. They identified two fundamental phenomena — goal hijacking and prompt leaking — that can occur when a language model is deployed in a dialogue setting. Greshake et al. [7] showed that the instruction does not have to originate directly in the prompt: an attacker can embed malicious code, data, or other text in a code repository, website, or e-mail, and the result will still be compromised if the model ingests it while performing searches. This latter finding is especially pertinent to the browsing agents of today's assistants.

Measuring agentic risk. Zhan et al. [3] evaluated the susceptibility of tool-using agents to coercive inputs by testing their ability to avoid content crafted by an attacker. They found that the outcome depended on the model and prompting strategy used. Liu et al. [4] demonstrated that there is a universal optimization process underlying these coercive prompts. Their finding suggests that many existing defenses, which are often evaluated on a limited set of manually crafted examples, could be ineffective against automatically generated, more sophisticated coercive inputs.

Defense and guidance. Hines et al. [5] proposed a number of spotlighting techniques that allow the model to distinguish between user instructions and some other text the user might have pasted by mistake. By evaluating their effectiveness on standard payloads, they showed that such modifications could significantly reduce an LLM's susceptibility to attacks in practice. OWASP [2] offers general guidance on improving the security of LLM applications, and Maloyan and Namiot [1] discuss attacks and defenses specific to code agents. Finally, Sahoo et al. [6] do not discuss adversarial inputs, but their survey of prompt engineering techniques is crucial background for understanding how the tool-use capabilities of modern assistants are implemented.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

IV. Research Gaps and Challenges

Considering Tables II and III together, we observe a pattern in which only one out of eight studies focuses on coding assistants and only one other study evaluates a defense against an attacker that adapts its approach (Fig. 1). None of the studies combine all three aspects (a benchmark, adaptive attack testing, and a tested defense) and the main reasons for this gap are discussed below.

Gap 1: There is limited evidence focused on coding assistants; most research evaluates general LLM applications or chat-based agents [8], [7], [3]. Maloyan and Namiot [1] directly evaluate coding assistants, but their work synthesizes prior research, rather than presenting novel experiments.

Gap 2: Evaluations against adaptive attackers are limited. Spotlighting and the OWASP guidance report useful results against non-adaptive payloads [5], [2]. Liu et al. [4] consider attacks that can be modified to bypass a defense, but their evaluation uses a small open source model and not a commercial coding assistant.

Gap 3: Attack benchmarks and defense techniques are rarely evaluated together. InjecAgent [3] evaluates response to indirect prompt injection attacks, but does not present a defense; Spotlighting [5] presents a defense technique, but is not evaluated on agentic-based tool-use attack. Thus, these two lines of work do not fully evaluate how well the proposed defense technique would respond to the attacks in practice.

Gap 4: A protocol-based defense is missing from the reviewed papers. While prior guidance recommends actions such as least privilege or human approval [2], [1], none of the eight papers implement and evaluate a layered approach encompassing source marking, adaptive testing, restricted tool access, and approval rules for an MCP-based coding agent.

V. Proposed Defense Framework

A. Risk Formulation

Based on the risk factors discussed in the reviewed guidance [2], we use the following simple expression to describe the risk in a deployment:

$$Impact \propto E \times A \tag{1}$$

Here, E is the extent to which the agent can access potentially untrusted content, while A is the exposure of the agent to files, the shell, and the network. It is rarely desirable to prevent an assistant from accessing external information, as that is often the point of having one. Rather, one should aim to restrict what an assistant can do with the information it finds. This is the guiding principle behind the following layers of protection.

B. Architecture

Figure 2 presents the proposed pipeline. It brings together five measures that already appear in the reviewed literature, rather than introducing a new defense technique.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

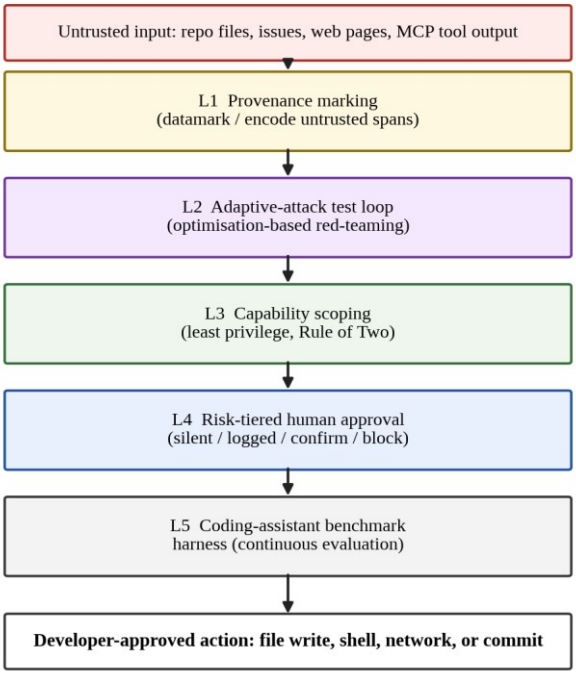


Fig. 2. The proposed five-layer defense pipeline. Layers L1 and L2 aim to reduce the chance of an attack succeeding; L3 to L5 are intended to limit the impact if an attack gets through.

L1: provenance marking. Before untrusted text is fed to the model, it is marked or encoded, using an approach such as spotlighting [5]. The aim is to assist the model to differentiate material that it is reading from instructions that it should follow.

L2: adaptive-attack testing. The system is tested, prior to deployment and at regular intervals, with attacks that have been adapted to evade defenses, following the general approach of Liu et al. [4]. This is critical, as testing only against a fixed set of prompts may give an overly optimistic assessment of the system's resilience.

L3: capability scoping. Each tool should only be granted the file, network, and shell permissions that it actually needs. This follows the least-privilege principle as recommended in [2] and [1] , and limits potential damage if the assistant is misled.

L4: risk-tiered human approval. Actions are categorised according to levels of risk or difficulty of reversal; the categories include allowed without interruption, logged, requiring confirmation, and blocked. By tiering according to risk, developers can focus their efforts on those decisions which require most attention [1].

L5: coding-assistant benchmark harness. The assistant is regularly tested against a suite of tool-use cases, inspired by InjecAgent [3]. This would link security testing with the proposed defenses, and identify if changes to the system's behaviour are actually improving its overall behaviour.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

VI. Proposed Evaluation Protocol

The framework has not yet been implemented, so this paper does not report experimental results. If it is built in future work, we suggest evaluating it using the following measures.

- The attack success rate before and after each defense layer is added, using both fixed and adaptive attack payloads.
- Benchmark coverage, measured by the proportion of cases in an InjecAgent-style test set that the system successfully blocks.
- Utility cost, measured by any change in normal task accuracy and the time the system takes to respond.
- Approval fatigue, measured by how often a developer approves an action without carefully reading the prompt or request.

Since the evaluation would not use classified or proprietary code, the first tests could use public repositories and benchmark cases that are openly available.

VII. Limitations and Scope

This review analyzes eight selected papers, and it should not be considered a systematic review of the entire body of work on prompt injection. The sources were selected for the seminar, and the ratings in Table III reflect the primary focus of each paper; a capability mentioned briefly might not be sufficiently represented. The five-layer pipeline is a proposal, not an implemented system. It lacks specific implementation details, such as particular thresholds or weights, and no experiments are presented to support performance claims. In addition, much of the measured evidence predates the rise of MCP, and the results might not directly apply to the current ecosystem.

VIII. Conclusion and Future Work

The eight studies identified demonstrate the progression of prompt injection attacks from a simple user instruction to the chatbot to automatically generated attacks against systems that employ tools. Altogether, the works identified illustrate language models' continued challenges in distinguishing between instructions and content. Finally, the review also underlines two key limitations in the current literature: the few studies that evaluate coding assistants directly and the lack of adaptive attacks. Additionally, those works reviewed do not combine attack benchmarking and a fully tested, multi-tier defense. This paper fills these gaps by compiling a three-level attack taxonomy, a simplistic risk assessment, and a five-tier defense pipeline based on mechanisms discussed in the literature.

The next step is to implement and test the proposed pipeline. Work could start with provenance marking and a benchmark harness followed by tests against adaptive attacks across the whole system, and the approval levels could be tweaked using the feedback from developers

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

References

1. N. Maloyan and D. Namiot, "Prompt injection attacks on agentic coding assistants: A systematic analysis of vulnerabilities in skills, tools, and protocol ecosystems," arXiv:2601.17548, Jan. 2026.
2. OWASP GenAI Security Project, "OWASP Top 10 for LLM Applications 2025," v2.0, 2025.
3. Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents," in *Findings Assoc. Comput. Linguistics: ACL 2024*, 2024, pp. 10471–10506.
4. X. Liu, Z. Yu, Y. Zhang, N. Zhang, and C. Xiao, "Automatic and universal prompt injection attacks against large language models," arXiv:2403.04957, Mar. 2024.
5. K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kıcıman, "Defending against indirect prompt injection attacks with spotlighting," arXiv:2403.14720, Mar. 2024.
6. P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," arXiv:2402.07927, 2024.
7. 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 *Proc. ACM Workshop Artif. Intell. Secur. (AISec)*, 2023, arXiv:2302.12173.
8. F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," in *Proc. NeurIPS ML Safety Workshop*, 2022, arXiv:2211.09527.
9. Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *Proc. 33rd USENIX Secur. Symp.*, 2024, arXiv:2310.12815.
10. E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," in *Adv. Neural Inf. Process. Syst. (NeurIPS), Datasets Benchmarks Track*, vol. 37, 2024, arXiv:2406.13352.
11. Q. Zhan, R. Fang, H. S. Panchal, and D. Kang, "Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents," in *Findings Assoc. Comput. Linguistics: NAACL 2025*, 2025, pp. 7101–7117.
12. E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The instruction hierarchy: Training LLMs to prioritize privileged instructions," arXiv:2404.13208, Apr. 2024.
13. S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: Defending against prompt injection with structured queries," in *Proc. 34th USENIX Secur. Symp.*, 2025, arXiv:2402.06363.
14. E. Debenedetti *et al.*, "Defeating prompt injections by design," arXiv:2503.18813, Mar. 2025.
15. J. Yi *et al.*, "Benchmarking and defending against indirect prompt injection attacks on large language models," in *Proc. ACM SIGKDD Conf. Knowl. Discovery Data Mining (KDD '25)*, Toronto, ON, Canada, 2025, arXiv:2312.14197.
16. X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (MCP): Landscape, security threats, and future research directions," arXiv:2503.23278, Mar. 2025.
17. Y. Liu *et al.*, "Prompt injection attack against LLM-integrated applications," arXiv:2306.05499, Jun. 2023.
18. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2022, arXiv:2108.09293.
    A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," arXiv:2307.15043, Jul. 2023.
19. S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023, arXiv:2210.03629.