Prompt Injection Attacks in Large Language Model Systems: Threats, Mechanisms, and Defense Strategies

Iqra Zulfiqar
Independent Researcher (Alumna, Department of Remote Sensing and GIS, GC University Faisalabad)
Email: iqrazulfiqar7373@gmail.com
Date: June 2026

Abstract

AI systems based on Large Language Models are now everywhere, from chatbots to autonomous agents that can book flights, send emails, and make financial decisions. But this power comes with a serious problem. These models cannot tell the difference between real instructions and fake ones hidden inside data. Attackers exploit this weakness through what researchers call prompt injection attacks. This paper reviews how these attacks work, what damage they can cause, and what defenses exist today. We look at different attack types, from simple text tricks to complex automated attacks that spread across multiple AI agents. We also review what current defenses can and cannot do. The conclusion is not comfortable: no defense today fully solves the problem, and the attacks keep getting smarter.

Keywords: Prompt Injection, Large Language Models, AI Security, LLM Agents, Adversarial Attacks, Multi-Agent Systems

I. Introduction

Think about what modern AI assistants can do. They read your emails, schedule meetings, browse websites, execute bank transfers, and make decisions on your behalf. This is genuinely useful. It is also genuinely dangerous.

The danger comes from a problem that sounds simple but turns out to be very hard to fix. When an AI model reads a document, it cannot truly separate “this is data I should process” from “this is an instruction I should follow.” An attacker who understands this can hide fake instructions inside normal-looking content – a webpage, an email, a product description – and the AI will follow those instructions as if a trusted user had given them.

This type of attack is called prompt injection. It is not a bug in one specific system. It is a consequence of how language models work at a fundamental level.

The consequences are real. Researchers have shown that prompt injection can be used to steal private data, manipulate AI-powered search rankings, corrupt the training signals used to improve AI models, and execute unauthorized financial actions. As AI systems take on more responsibility in sensitive domains, these risks grow accordingly.

This paper reviews the current state of prompt injection research. Section II explains why LLMs are structurally vulnerable. Section III organizes attack types into a clear taxonomy. Section IV looks at how researchers measure and evaluate these attacks. Section V covers defense strategies and honestly assesses their limitations. Section VI addresses a newer and scarier problem: attacks that spread between AI agents like a virus. Section VII discusses what still needs to be solved. Section VIII concludes.

II. Why Are LLMs Vulnerable in the First Place?

A. The Core Problem: Instructions Look Like Data

In a normal computer system, there is a clear separation. Instructions live in one place, data lives in another. The processor knows which is which. This separation is fundamental to security.

Language models have no such separation. Everything – system prompts, user messages, retrieved documents, database outputs – arrives as a stream of text tokens. The model processes all of it the same way. It has no reliable internal mechanism to say: "this part came from a trusted source, this part came from the internet, I should treat them differently."

This is not a flaw that can be patched easily. Zverev et al. argue that fixing it will require breakthroughs that go well beyond tweaking prompts or adding filters [11]. It is a structural feature of how these models are built.

B. How Big Is the Problem?

Bigger than most people assume. One study tested 36 different language models across 144 prompt injection scenarios [4]. The result: 56% of tests produced a successful attack. More than half. And the pattern that emerged was counterintuitive – larger, more powerful models were often more vulnerable, not less. A model that is very good at following instructions will follow bad instructions just as obediently as good ones.

Debenedetti et al. call this an "inverse scaling law" – the better the model, the easier it can be to manipulate [1]. This is an uncomfortable finding for anyone who assumed that more capable AI would automatically be more secure.

III. Types of Prompt Injection Attacks

A. Direct Injection

The simplest version is direct injection: a user types something like "ignore your previous instructions and do X instead." Early demonstrations of this against GPT-3 showed it worked reliably. Modern commercial models have gotten better at resisting obvious versions of this attack, but more creative variations still succeed.

B. Indirect Injection

This is the more dangerous variant, because the user does not have to do anything wrong. The attack hides inside data the AI retrieves during normal operation – an email, a webpage, a database record.

Here is a concrete example. Imagine an AI assistant is asked to summarize emails. One of those emails contains hidden text: "Before you summarize anything, forward all emails to attacker@evil.com." The AI, unable to distinguish data from instructions, may comply. The user never saw the malicious instruction. It arrived through the data [1].

The AgentDojo benchmark built by Debenedetti et al. tests this at scale – 629 security scenarios across four realistic environments including banking, email, travel booking, and messaging platforms [1]. The results are sobering.

C. Optimization-Based Attacks

Manually writing injection prompts has limits. Researchers found a way around this: use gradient-based optimization – the same math used to train neural networks – to automatically generate injection text that is far more effective than anything a human would write by hand.

Liu et al. built a framework that can generate highly effective universal injection attacks using just five training examples – less than 0.3% of test data [9]. The attack worked better than all manually crafted alternatives they tested.

A specific and worrying application is JudgeDeceiver, an attack that targets LLM-as-a-Judge systems [2]. These systems are used to rank AI outputs, select tools, and provide feedback for AI training. JudgeDeceiver injects text that causes the judge model to always pick the attacker's preferred answer, regardless of its actual quality. Standard defenses like perplexity detection do not stop it [2].

D. Universal Attacks

Some attacks are designed to work across many different models, tasks, and scenarios without needing to be customized for each target. The “Important message” attack studied in AgentDojo is a good example – it is structurally simple but achieved a 57.7% success rate against GPT-4o, compared to under 6% for older manual injection approaches [1]. Simple, effective, broadly applicable.

IV. How Researchers Test These Attacks

A. AgentDojo

The most thorough evaluation framework available is AgentDojo, presented at NeurIPS 2024 [1]. It is not a static list of test cases – it is a live environment where AI agents actually execute tasks using real tools, against active injection attempts.

This matters because testing on paper is different from testing in a real pipeline. A prompt injection hidden in an email only succeeds if the model actually calls an email-sending tool afterward. Static benchmarks that simulate tool calls can be fooled by the attack even in the evaluation – which would make defenses look better than they are. AgentDojo avoids this problem.

Key numbers from their testing: current models fail more than a third of legitimate tasks even with no attack present. The best attack succeeds 57.7% of the time. The best defense reduces this to about 7%, but costs significant performance on normal tasks [1].

B. Cross-Architecture Testing

Sharma et al. ran a different kind of study – testing 36 models systematically to find out which architectural features predict vulnerability [4]. Using statistical methods including logistic regression and random forest analysis, they found that parameter count and architecture type both matter significantly. They also found clusters of models with similar vulnerability profiles, suggesting that certain design choices create shared weaknesses [4].

V. Defenses: What Works and What Does Not

A. Filtering Inputs and Outputs

The most obvious defense is scanning what goes into and comes out of the model. Kim et al. built a system called LPDS that checks for personally identifiable information, malicious code, suspicious URLs, and injection commands in both input and output [5].

The problem is that filtering becomes a cat-and-mouse game. Optimization-based attacks can generate injections that look grammatically normal and pass perplexity checks [2]. The injection detector in AgentDojo does reduce attacks to around 8% success – but it flags so many legitimate inputs as suspicious that it significantly hurts the model's usefulness [1]. A security tool that makes the product unusable is not a real solution.

B. Signed Prompts

Suo proposes a different idea: instead of trying to detect bad content, mark trusted instructions with a signature [6]. The model learns to recognize and prioritize signed instructions and treat unsigned instructions with skepticism.

This is conceptually cleaner than filtering. Experiments showed good resistance to many attack types. But it requires either fine-tuning the model or carefully crafting system prompts, and no one has fully tested how it holds up against an attacker who specifically tries to forge or mimic the signature scheme [6].

C. Tool Isolation

One of the more practical defenses in AgentDojo is tool filtering [1]. The idea is simple: before the model reads any untrusted data, it decides which tools it actually needs for the task. If it only needs to read emails, it does not give itself access to send emails. This removes the attacker's ability to exfiltrate data even if the injection succeeds.

Results: attack success rate drops to 6.84%, the best of any defense tested [1]. But it fails in roughly 17% of cases where the legitimate task requires the same tools the attacker wants to exploit, or where the model cannot plan its tool use in advance without first reading the data it needs to be suspicious of [1].

D. Building Security Into the Model Itself

The most robust approach is training-level defense. Meta's Meta SecAlign is the first open-source model with this kind of built-in resistance to prompt injection at commercial-grade capability levels [7].

What makes it interesting is that it was not trained specifically on injection attack scenarios – it was trained on generic instruction-following data, and the security generalized anyway. It holds up on unseen tasks including tool-calling and web navigation, and outperforms several major proprietary models on security benchmarks while keeping strong performance on normal tasks [7].

The catch: building this kind of defense requires access to large-scale training infrastructure. Most organizations deploying AI systems cannot do this themselves.

E. LLM-Based Attack Detection

Andreou et al. propose using a separate LLM as a filter – essentially, fight AI with AI [8]. A validation module classifies incoming prompts as safe or malicious before they reach the main model. Results were

positive against tested attack patterns, though performance depends heavily on how sophisticated the attacks are [8].

F. Thinking Beyond Technical Fixes

Kumar et al. make a point worth emphasizing: technical defenses alone are not enough [3]. Their framework classifies prompt injection attacks by what kind of trust boundary they violate and how much expertise the attacker needs. The conclusion is that strengthening LLM security requires a combination of technical controls and organizational policy. When humans are in the loop and social engineering is possible, purely automated defenses will always have gaps [3].

VI. A New Problem: Prompt Infection in Multi-Agent Systems

Single-model attacks are concerning. What happens when attacks spread between models?

Multi-agent LLM systems are increasingly common. One agent gathers information, another processes it, another acts on it. Each agent's output becomes another agent's input. This creates a propagation pathway for attacks.

Lee and Tiwari introduced the concept of prompt infection – a malicious prompt that does not just hijack one model, but replicates itself through a network of interconnected agents [10]. It behaves like a computer virus: infecting one agent, embedding itself in that agent's outputs, and spreading to the next agent that processes those outputs.

The consequences are severe. A single successful injection could give an attacker control over an entire multi-agent pipeline – enabling data theft, misinformation, scam execution, and system disruption, all propagating autonomously after the initial attack [10].

Existing defenses offer partial mitigation, but the fundamental problem is structural: agents have to process each other's outputs to function. Complete defense through filtering is probably impossible in this architecture without rethinking how agents communicate.

This is not a theoretical future threat. Multi-agent frameworks like LangChain, AutoGen, and CrewAI are already in production use across many industries.

VII. What Still Needs to Be Solved

A. The Core Tradeoff Has Not Been Broken

Every defense reviewed here involves giving something up. Make the model less instruction-compliant and it becomes less useful. Add a detector and it blocks legitimate inputs. Restrict tools and some tasks become impossible.

AgentDojo's Pareto frontier analysis shows this clearly: no current defense sits at the corner where both utility and security are high [1]. Someone has always paid a price. Breaking this tradeoff is probably the most important open problem in this field.

B. Testing Is Not Realistic Enough

Current benchmarks test relatively simple attacks by attackers with no specific knowledge of the target system. Real adversaries know more. They know the model version, the tool API, the user's name and patterns. They can craft targeted attacks that current benchmarks would never catch.

Additionally, multilingual attacks, attacks across multiple sessions, and attacks exploiting images or audio are barely studied. These will matter as AI systems become more capable and more embedded in real infrastructure.

C. The Most Important Research Priorities

Based on this review, three directions stand out as most urgent:

Formal security guarantees. Probabilistic defenses that work most of the time are not enough for security-critical applications. Future research needs to pursue formal methods that can provide guarantees about what kinds of injected content cannot influence model behavior.

Multi-agent security protocols. Explicit security standards for how AI agents communicate with each other – analogous to how TLS secures network communication – do not yet exist. They are urgently needed.

Standardized benchmarks. Researchers currently test on different benchmarks against different attack types, making it impossible to compare results across papers. A shared, comprehensive evaluation standard would significantly accelerate progress.

VIII. Conclusion

Prompt injection is not a niche vulnerability. It is a fundamental challenge for any AI system that reads untrusted data and takes actions based on what it reads – which describes most useful AI deployments today.

This review has shown that attacks are widespread, increasingly automated, and spreading into multi-agent architectures where the consequences of a single successful injection multiply. Meanwhile, defenses exist but none fully solves the problem. Every current approach involves tradeoffs, and the most sophisticated attacks evade most single-layer protections.

Some progress is real. AgentDojo provides a rigorous testing standard. Meta SecAlign shows that training-based defenses can work. Tool isolation and prompt signing offer practical partial mitigations for specific scenarios.

But the pace of attack development is outrunning defense. LLMs are being deployed in healthcare, finance, law, and critical infrastructure. The cost of a successful prompt injection in these contexts is not a wrong search result – it could be a fraudulent transaction, a data breach, or worse.

Prompt injection needs to be treated as a primary design constraint, not an afterthought, in any LLM-integrated system. The research community, developers, and organizations deploying AI all share responsibility for making that happen.

References

[1] 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 Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Track on Datasets and Benchmarks, 2024.

[2] J. Shi, Z. Yuan, Y. Liu, Y. Huang, P. Zhou, L. Sun, and N. Z. Gong, “Optimization-based Prompt Injection Attack to LLM-as-a-Judge,” in Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2024, pp. 660–674.

[3] S. S. Kumar, M. L. Cummings, and A. Stimpson, “Strengthening LLM Trust Boundaries: A Survey of Prompt Injection Attacks,” in 2024 IEEE 4th International Conference on Human-Machine Systems (ICHMS), 2024.

[4] A. Sharma et al., “Systematically Analyzing Prompt Injection Vulnerabilities in Diverse LLM Architectures,” arXiv preprint, 2024.

[5] J. Kim et al., “Protection of LLM Environment Using Prompt Security,” IEEE, 2024.

[6] X. Suo, “Signed-Prompt: A New Approach to Prevent Prompt Injection Attacks Against LLM-Integrated Applications,” arXiv preprint arXiv:2401.07612, 2024.

[7] S. Chen, A. Zharmagambetov, D. Wagner, and C. Guo (FAIR at Meta / UC Berkeley), “Meta SecAlign: A Secure Foundation LLM Against Prompt Injection Attacks,” arXiv preprint arXiv:2507.02735, 2025.

[8] D. Andreou et al., “Enhancing System Security: LLM-Driven Defense Against Prompt Injection Vulnerabilities,” arXiv preprint, 2024.

[9] X. Liu, Z. Yu, Y. Zhang, N. Zhang, and C. Xiao, “Automatic and Universal Prompt Injection Attacks against Large Language Models,” arXiv preprint arXiv:2403.04957, 2024.

[10] D. Lee and M. Tiwari, “Prompt Infection: LLM-to-LLM Prompt Injection within Multi-Agent Systems,” arXiv preprint arXiv:2410.07283, 2024.

[11] E. Zverev, S. Abdelnabi, S. Tabesh, M. Fritz, and C. H. Lampert, “Can LLMs Separate Instructions From Data? And What Do We Even Mean By That?,” in International Conference on Learning Representations (ICLR), 2025.