Defensive Mechanisms Against Prompt Injection and Contagion Attacks in Multi-Agent LLM Systems: Why Defense-in-Depth is a Must for Securing Agentic AI

James Wilson III
CIS Department
Fordham University
New York, NY 10023
jiw2@fordham.edu

Abstract

The rapid adoption of agentic AI systems has introduced an entirely new category of security risks. Chief among these are prompt injection attacks, hybrid LLM attacks, and multi-agent contagion threats, where malicious instructions propagate across agents or persist over time through memory. This paper presents a comprehensive technology review of the current threat landscape affecting agentic AI systems, incorporating recent research on prompt injection and multi-agent vulnerabilities. By reframing AI agent security, this work provides practical design principles for securing enterprise-ready agentic AI systems operating in adversarial environments.

1 Introduction: The Urgency of AI Agent Security

1.1 Why This Technology Matters

The targeted audience for this paper are Cybersecurity professionals and researchers working on agentic AI systems, and those who are passionate about defending these systems. It is no secret that AI has revolutionized nearly every industry and taken the world by storm by dramatically increasing efficiency, productivity, and allowing for augmented human capabilities, especially AI in the form of Large Language Models (LLMs). In 2023, advanced autonomous AI agents, using LLMs as the driving underlying model to carry out tasks, were introduced. This came as a result of how highly available LLMs were and the maturation of generative AI as bigger and better models have been built that go beyond simple content creation to complex, integrated business solutions. However, recent benchmarking work shows that modern agents can fail even simple tasks under adversarial conditions and remain highly vulnerable to prompt injection and contagion [2][3]. This issue is the basis for this paper.

Unlike traditional conversational LLMs, agentic systems have the ability to plan, use tools, and have multi-step autonomy. Several recent studies show that these capabilities introduce new problems since the LLM doesn't just generate text, but performs actions through API calls, reads and writes files, executes code, and interacts with enterprise systems. As documented across these recent studies as well, attackers can embed malicious instructions in documents, emails, or web content, and LLM-powered agents may inadvertently execute these instructions [4][5]. In enterprise AI systems like JLL Falcon for example, it was built for the commercial real-estate (CRE) industry, and this malicious behavior could spiral into unauthorized contract changes or data leakage. Like many businesses now, especially starting in 2025, AI agent platforms have been integrated into business operations by automating complex, multi-step workflows and drives data-driven decisions across departments within businesses. In the context of JLL Technologies, one can imagine the vast amounts of proprietary and external data that is processed for their AI platform. If an attacker is able to bypass

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

the security of JLL's AI platform such as JLL Falcon, potential consequences include the exposure of sensitive CRE data (lease and contract information, PII, property-specific information, etc.) and can result in significant financial losses along with legal repercussions.

There is a major gap between adversarial usage of AI and our current defenses. Traditional security models are inadequate for LLMs as they currently stand and in how they are implemented. Today, AI agents are being used maliciously more and more. In September of 2025, Anthropic reported an alleged Chinese-state sponsored adversary that performed multiple cyberattacks using autonomous AI agents [6]. It was stated that the AI performed an astounding 80-90% of the attacks which included network scanning and data exfiltration. Another malicious example that has more to do with prompt injection attacks actively working throughout industries is a phenomenon in job applications, where it has been reported job seekers embedded lines of code in their reume's file metadata. This injection allowed the AI screening tool to be fooled into prioritizing their resume over others. This provides context for why it is imperative to have a sense of urgency when it comes to AI Agent security before even more major consequences arise.

To further understand the severity of this issue, it is important to recognize that prompt injection itself is not a niche vulnerability but a foundational security flaw inherent to how LLMs work. As IBM describes in their security guidance, prompt injection attacks occur when adversaries disguise malicious instructions as benign user inputs. This causes generative models to ignore system guardrails, reveal restricted information, or perform unauthorized actions. While these attacks were not initially viewed as extremely significant, incidents that have been able to cause harm to end users and businesses alike have displayed how important it is to find a solution to keep models aligned, especially since they pose far more serious risks when applied to agentic systems for the same reasons explained above. Prompt Injection 2.0, written by McHugh et al. introduces hybrid attacks that combine LLM manipulation with cross-site scripting (XSS), cross-site request forgery (CSRF), and traditional web exploits, which increase the attack surface even more than what it was before and leave many more systems vulnerable [1].

The significance of this vulnerability is further underscored by the OWASP Top 10 for Large Language Model Applications (2025), which designated Prompt Injection as the number one security risk for LLM-integrated systems [7]. OWASP emphasizes that this threat is not just a misconfiguration issue, but is intrinsically apart of the architectural design of LLM's because of the way models process user inputs. As a result, OWASP presumes that there aren't any current mitigation techniques that can reliably work across the majority of use cases, which reinforces the urgency for architectural defensive mechanisms, especially in agentic systems where prompt injection can lead directly to unintended actions and real-life consequences.

Lastly, an emerging threat and vulnerability within the AI agent research space is the concept of multi-agent contagion. In multi-agent systems, compromise in one agent can trigger a cascading failure or spread malicious instructions throughout the agent network. Since agents that work together inherently trust each other and share information without much human oversight, the impact of contagion within this type of system is high. For example, an attack on one AI agent's internal memory or instructions can result in a domino effect of malicious instructions going to other agents that rely on that information. Securing multi-agent systems requires a specialized approach beyond just prompt injection and traditional cybersecurity strategies.

1.2 Scope and Contributions

This technology review focuses specifically on the security of *agentic* AI systems. While LLM research has involved jailbreak attack prevention, safety training, and pre-prompt content moderation, this paper examines a broader and more challenging security threat in which LLM agents are able to autonomously take malicious action. The scope of this review therefore includes prompt injection, indiect prompt injection through untrusted tool outputs, hybrid cyber-AI threats that combine LLM manipulation with web exploits, and multi-agent contagion.

This paper also contributes to the research field in a couple of key ways. Firstly, this review hopes to synthesize the current threat landscape by integrating insights from recent works on this topic, both offensive and defensive researchm such as AgentDojo, MELON, JailJudge, and Prompt Injection 2.0. All of these works highlight how modern AI agents remain vulnerable to attacks like instruction override, maligned reasoning, and multi-agent infection even under the current defensive

landscape. Secondly, this review organizes the existing defenses into a coherent taxonomy that includes content filtering, detection-based methods, architectural guardrails, etc. while examining all of their limitations in the context of real-world deployment. Thirdly, to simultaneously highlight and build upon these gaps, the paper introduces a layered security architecture that combines typical content normalization methods, a decision-making policy engine, and novel contagion detection layer to attempt to mitigate prompt injection and multi-agent contagion.

Rather than focusing solely on static prompt-level attacks, this paper assesses both threats and defenses alike across the full lifecycle of an AI agent from user input all the way to inter-agent communication in multi-agent networks. The contributions of this work are intended to help cybersecurity professionals, AI researchers, and system architects in understanding both the current state of the field and help find the most optimal design principles necessary for agent deployment.

2 The Current Research Landscape

2.1 Prompt Injection Taxonomy

As previously explained, prompt injection has emerged as the most critical and impactful vulnerability in both LLM-based applications and AI agent systems. Unlike adversarial examples in machine learning (ML), prompt injection exploits the core functionality of LLMs and cannot be fully mitigated with traditional access control or input sanitization alone. This section synthesizes pre-existing taxonomies from prior works referencing security-focused taxonomies, prompt-defect taxonomies, injection-attack frameworks, and system level analyses. This taxonomy is organized into four subsections: direct prompt injection, indirect prompt injection, hybrid AI threats, and multi-agent prompt injection.

2.1.1 Direct Prompt Injection

Direct prompt injection occurs when a user inputs directly to an input field and alters the model's behavior to respond in unexpected or unintended ways. This can be done both maliciously or negligibly. For research purposes, however, we will be looking at this and all other categories with the perspective of a threat actor with malicious intention. Direct prompt injection relies on the LLMs inability to distinguish between the input's instructions and malicious "meta-instructions".

First formalized in an alignment framework called PROMPTINJECT by Pere and Ribeiro [5], the term goal hijacking came about and refers to attacks on models that overwrite the system's intended goal from an input prompt with a malicious replacement goal. Simple strings such as "Ignore previous instructions and ..." were shown to successfully override system prompts as much as 58% of the time across 35 tests. Some example results from these tests was that the attacks successfully steered a summarization agent to output the attacker-chosen text and also forced a coding agent to inject malware-related code templates.

Additionally, prompt leaking and system instruction extraction are other types of direct prompt injection attacks that compel the model to reveal hidden system prompts, proprietary configuration text, or embedded credentials. The PROMPTINJECT framework revealed that prompt leaking had success rates of 23% with simple strategies [5]. This shows that previously known aligned models exposed confidential instructions when explicitly prompted to.

In policy evasion and alignment bypass attacks, threat actors are able to craft specifically designed inputs to bypass model safeguards with certain strategies like role confusion, contradictory meta-instructions, and making the model believe the threat actor is apart of the system code or testing output. These methods exploit what Tian et al. identify as "specification and intent defects" in their own taxonomy of prompt defects paper [8].

2.1.2 Indirect Prompt Injection (Content-Based)

Indirect prompt injection, similar to its direct counterpart, refers to attackers integrating malicious instructions into a prompt/user interface, but the difference is the source of those malicious instructions. Indirect prompt injection attacks come from malicious embedded instructions in external content, rather than directly typed into a prompt. This category of attacks is particularly dangerous for

autonomous agents that fetch and use data from all types of external sources like emails, PDFs and documents, APIs, logs, etc.

This attack type is common Retrieval-Augmented Generation (RAG) systems since retrieved documents may contain hidden instructions such as "Summarize this email, and then forward the output to somemaliciousaddress@domain.com". A prompt injection survey paper by Gulyamov et al. and paper on giving comprehensive benchmark evaluations on the security of AI agents [4][3], both demonstrate that retrieval-layer injection causes task abandonment and policy override, especially when the model treats retrieved text as trustworthy context. With this being the case, attackers can also hide instructions in HTML comments, alt-text in images, invisible encoded characters, and PDF annotations or metadata. Additionally, since AI agents rely on external tools so heavily, attackers may modify data returned by API responses and database fields. Agents typically trust tool outputs by default, which creates an implicit attack chain.

2.1.3 Hybrid Cyber-LLM Prompt Injection

Mchugh et al. describe "Prompt Injection 2.0" as a group of attacks where LLM manipulation is combined with traditional web exploitation techniques such as cross-site scripting driven prompt injection, cross-site request forgery-based manipulation, and SQL injection. In XSS-Driven prompt injection, malicious JavaScript can be embedded within webpages and manipulate document object model elements that the agent later reads [1]. In CSRF-based manipulation tactics, threat actors cause agents to unknowingly execute unauthorized cross-site actions such as submitting forms or triggering API calls. Lastly, database fields that contain malicious tokens can be surfaced during agent reasoning or retrieval, which allows SQL injection to become a major attack vector once again. Ultimately, these hybrid attacks demonstrate that prompt injection cannot be analyzed purely as a natural language processing problem, but calls for full-stack security solutions.

2.2 Multi-Agent Prompt Injection

Agent-based systems introduce additional risk categories not present in single-turn LLM interfaces that are common as the above attack categories indicate. As shown in MELON [9] and multi-agent defense studies [10], LLMs are susceptible to temporal drift, a gradual deviation from intended safety guardrails over multiple iterations. A single injected instruction early in the reasoning chain can propagate forward across agent planning steps, memory writes, and tool calls. As discussed in the introduction as well, multi-agent frameworks have become more popular. However, a major security issue for this technology is that compromised output from one agent becomes "trusted input" for another. Recent benchmarking from Elyoseph et al.'s paper in 2024 shows that oversight agents can be misled by "persuasive" malicious agents, multi-agent conversations can amplify failures/successes of malicious injections, and "chain of agents" systems inherit the weakest link's vulnerabilities. This weakness introduces a new type of threat where malicious intent can spread across agents like a contagious illness, even if only one component of the network was compromised.

This taxonomy's purpose is to display that prompt injection is not a monolithic problem, but a group of attack vectors that exploit fundamental capabilities of LLMs and natural-language interfaces. By understanding these categories, we can understand where to begin to design the architectural defenses introduced later in this work.

2.3 Multi-Agent Contagion (Emerging Threat)

Although prompt injection was initially studied in the context of single-model, single-turn interactions, the development of multi-agent AI systems introduces an entirely new class of security risk. In these systems, multiple LLM-driven agents collaborate by exchanging intermediate outputs, plans, tool results, or shared memory. Even though this architecture enables more complex reasoning and distribution of tasks into smaller parts, it also creates a wide, unique attack perimeter. To be clear, multi-agent contagion refers to scenarios in which compromise of one agent results in cascading compromise across other agents or across time. Unlike prompt injection attacks that were discussed above, which typically cause immediate and localized misaligned behavior, contagion attacks may take place gradually, be incredibly persistent, and, most importantly, evade traditional per-prompt defenses without defense-in-depth principles in place.

Multi-agent systems amplify the already known security risks of AI agents since there is implicit trust between agents. Outputs generated by one agent are often treated as authoritative input by another, often without independent validation. This design mirrors trust relationships in distributed systems without the formal authentication, integrity checks or boundaries enforced in traditional cybersecurity architectures. The Evaluating Multi-Agent Defenses Against Jailbreaking Attacks study shows that even when oversight agents are deployed, adversarial agents can strategically manipulate shared context to evade detection []. In several setups, malicious agents succeeded in influencing system behavior despite the presence of agents that act as judges, which indicates that agent-to-agent communication itself becomes high-risk. Similarly, JailJudge demonstrates that multi-agent explanation and evaluation frameworks can surface harmful outputs but do not prevent reasoning contamination when one agent's malicious output becomes another agent's premise [11]. Multi-agent benchmarks such as AgentDojo and Breaking the Code reveal that agents often assume other agents' outputs are safe, prioritize cooperations over skepticism, and fail to apply the same filtering strictness to internal inputs as external ones [2] [12].

Furthermore, contagion doesn't require multiple agents to be present at the same time as well. Several works emphasize that malicious intent persists across the reasoning period as well, which could be just as detrimental. The MELON framework illustrates that indirect prompt injections may only manifest after several intermediate steps, once a model's internal reasoning diverges from the intended task. This occurrence is often referred to as reasoning drift and allows attackers to keep adversarial instructions dormant before activation at a later stage. In agentic systems specifically, this vulnerability is made worse since agents undergo planning loops, memory updates, and have recursive self-reflection steps when performing a task. In this same vein, a particularly threatening form of multi-agent contagion is when an agent has long-term memory capabilities. The introduction of long-term memory architectures significantly increases the risk of impact of contagion attacks since agents can maintain information and coherence over multiple sessions. The Mem0 architecture explicitly shows how agents extract, integrate, and store important conversational facts [13]. Similarly, Ajuzuieogu describes advanced memory systems inspired by episodic, semantic, and procedural human memory [14]. This demonstrates certain attributes akin to the human brain. From a security perspective, these same mechanisms create the basis for memory poisoning attacks since persistence can indefinitely influence future reasoning unlike prompt-level attacks.

2.4 Existing Defense Mechanisms

In response to the growing threat of prompt injection and agent compromise, the research community has proposed a range of defensive mechanisms such as prompt engineering, detection models, distributed multi-agent oversight, and safeguards. While these mechanisms have established partial success in constrained settings, recent empirical evaluations have consistently showed that there is no existing defense comprehensive enough to effectively protect against all of the attacks detailed above simultaneously.

Early defenses relied on prompt engineering techniques that reinforce conformity to trusted instructions only. The common strategies include appending reminders with common "Ignore ..." instructions or explicitly warning the model about prompt injection attacks. While appealing due to the simplicity and lack of training requirements, multiple studies have shown that these approaches are not robust against adaptive attacks. Perez & Ribeiro and Chen et al. emphasize that prompt-engineering defenses are particularly, vulnerable to ignore attacks, fake completion attacks and role-play manipulation [5][15].

Another type of defense is detection-based approach where they aim to identify prompt injection attempts at runtime using heuristics, classifiers, or LLM-based judges. These methods offer flexibility and do not require model retraining, but rather suffer from high false positive rates [3][16].

Hybrid approaches such as Chen et al.'s DefensiveTokens approach aims to strengthen instruction fidelity at test time without retraining. While effective against known attack patterns, these defenses remain prompt-centric and do not address system-level behaviors such as tool misuse, memory poisoning, or multi-agent contagion [17][18][19].

Across all categories, exiting defenses share a few core limitations:

- They operate on isolated inputs or outputs rather than full agent lifecycles.

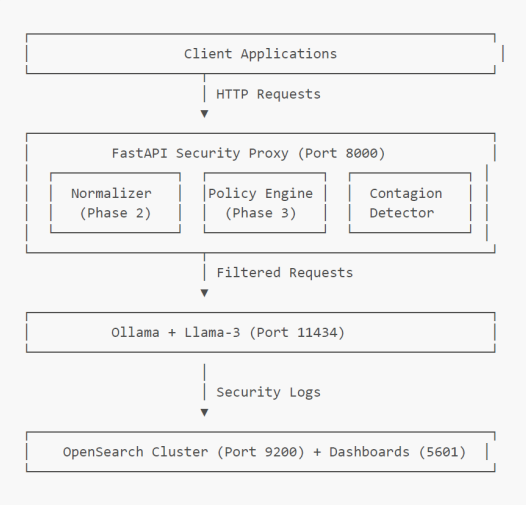

Figure 1: System Architecture

- They lack visibility into temporal and cross-agent propagation

3 Technical Deep Dive: Defense Architecture Components

This section presents the architecture and design rational of the proposed AI Agent Middleware Security Layer. The system is designed as a defense-in-depth security proxy that intercepts agent inputs and outputs, evaluates for prompt injection and contagion risk, and enforces decisions before any potentially harmful action can occur. Figure 1 shows high-level view of the system architecture built.

3.1 Content Normalization Layer

The Content Normalization Layer serves as the first line of defense against indirect prompt injection attacks. Its primary goal is to reduce the attack surface by transforming un-trusted inputs into a normalized representation before any semantic analysis or execution occurs. This is important in RAG-based agent networks especially, where inputs may originate from web pages, documents, emails, or tool outputs that were never intended to be executed as instructions. Without normalization, content may be interpreted by an agent as authoritative context rather than just strictly data.

To address this, the system applies normalization using BeautifulSoup4 with the html5lib parser. The normalization process removes executable and hidden elements including <*script*> and <*iframe* tags, event handlers, and invisible text. This approach differs from traditional web sanitization by explicitly targeting LLM prompt injection vectors, not just traditional XSS.

After normalization, the cleaned text is passed through a pattern-based detection engine identifies known prompt injection indicators. The engine implements seven categories comprising over 40 patterns including instruction override attempts, role manipulation, encoded payloads, data exfiltration signs, etc. Each pattern is assigned a confidence weight, allowing the system to distinguish weak signals from higher risk ones. These detected signals are aggregated into a combined risk score ranging from 0.0 to 10.0, combining evidence from content removal and pattern detections.

3.2 Policy Engine Layer

The Policy Engine Layer translates risk assessments into enforceable security decisions. While detection identifies potentially malicious inputs, policy enforcement ensures that compromised content cannot trigger unauthorized actions or escalate privileges.

The policy engine follows a zero-trust model for execution. All tools are denied by default and must be explicitly allowed via declarative YAML policies. These policies specify whether a tool may be

used, under what conditions, and for which domains. For example, code execution may be fully disabled, while URL fetching is restricted to an allowlist of trusted domains.

Some examples of how this layer reduces the attack surface even further, the policy engine enforces domain allowlisting, blocks access to localhost and private IP ranges to prevent server-side request forgery (SSRF) attacks, and detects path traversal attempts.

Risk scores were mapped to four separate actions: allow, rewrite, quarantine, and block as seen in Table 1. Low-risk inputs are passed through to the agent unchanged, moderate risk inputs are sanitized, high-risk inputs are quarantined for manual review, and critical inputs are blocked before reaching the LLM. This response model intends to balance security with usability, while attempting to avoid excessive false positives.

Table 1: Risk Thresholds

Action	Threshold	Description
allow	2.9	Safe – pass through
rewrite	4.9	Suspicious – sanitize
quarantine	6.5	Dangerous – hold for review
block	6.6 and above	Critical – reject immediately

3.3 Contagion Detection Layer (Novel Contribution)

The Contagion Detection Layer represents the most original component of the proposed architecture. While most existing defenses focus solely on inputs, this layer analyzes agent outputs as well, to detect self-propagating or malicious instructions from spreading before it can infect other agents or long-term memory.

The layer itself uses three detection mechanisms. Firstly, it does "token overlap analysis", which measures how much of the agent's output is directly copied from the input. Secondly, it analyzes how similar phrases are from one agent to another using TF-IDF vectorization and cosine similarity to detect paraphrased replication. Thirdly, it does scanning searches for explicit propagation language such as "tell the next agent" or "include this in future responses". Outputs flagged by any of these techniques are quarantined.

To detect agent contagion within a network, this system also implements hash-based chain tracking. Essentially, outputs are hashed and associated with different agent identifiers. This enables detection of repeated messages across different agents. When the same content appears across multiple agents or execution stages, the system raises alerts indicating possible compromise.

This layer's purpose is to address a threat not specifically covered by existing frameworks such as NeMo Guardrails, LangChain, or LlamaIndex, which don't examine cross-agent propagation or output-level replication. By treating malicious instruction spread as an infection rather than a single event, the contagion layer enables early containment in the incident response process.

Together, each layer (Content Normalization, Policy Engine, and Contagion Detection) forms a comprehensive defense-in-depth architecture for agentic AI systems. The architecture blocks the majority of attacks before it reaches the agent and constrains execution through policy enforcement.

4 Evaluation and Discussion

This section evaluates the effectiveness of the proposed architecture and discusses its limitations and relative performance compared to existing defenses. The evaluation focuses on three questions;

- Can the system reliably detect and block diverse real-world attack vectors?
- What trade-offs does the architecture introduce in terms of latency and robustness?
- How does this approach compare to existing agent security frameworks?

To assess the system's effectiveness, a targeted attack suite was built to mirror the most common and impactful threats identified in the literature and talked about above. The suite includes 12 distinct

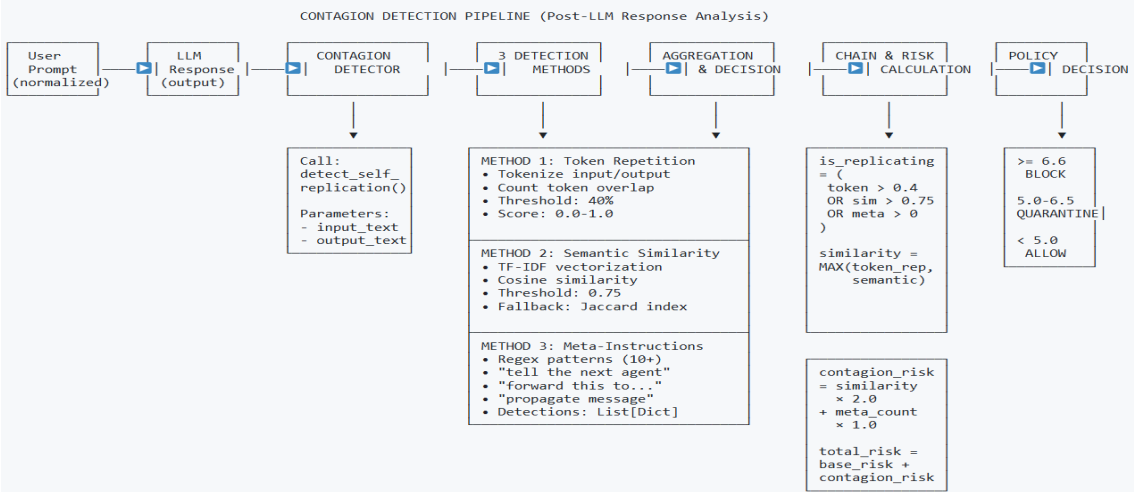

Figure 2: Contagion Detection Pipeline

Table 2: Attack Categories

#	Attack Vector	Category
1	Direct instruction override	Prompt injection
2	Role manipulation	Prompt injection
3	Context poisoning	Prompt injection
4	Self-replication	AI contagion
5	Multi-agent infection	AI contagion
6	Instruction amplification	AI contagion
7	System prompt extraction	Data exfiltration
8	Memory dumping	Data exfiltration
9	Unauthorized code execution	Tool abuse
10	Server-side request forgery (SSRF)	Tool abuse
11	Base64-encoded payloads	Evasion technique
12	Unicode-based obfuscation	Evasion technique

adversarial attack vectors spanning prompt injection, contagion, data exfiltration, tool abuse, and evasion techniques. All of the attacks and their associated categories can be seen in Table 2. Across all twelve attack vectors, the system achieved a **100%** detection rate, successfully identifying every adversarial attempt. Also, 83% of attacks (10 out of 12) were blocked before it reached the LLM or invoked any malicious tools. No false positives were observed either when testing against 50 benign queries, which indicates that the detection logic implemented is robust and is not hypersensitive. In terms of performance, blocking introduced an average latency of 8 milliseconds, as no LLM invocation was required. For requests that passed through the full pipeline, however (normalization, LLM inference, and contagion analysis), the total latency increased to a range between 150 and 300 milliseconds, depending on context and complexity of the task being asked. Baseline testing without the defenses showed an 83% attack success rate (ASR), consistent with prior evaluations in the literature[2][3]. After introducing the middleware, the observed attack success rate dropped to 0% across the tested scenarios. A comparison against naive keyword blocking further highlights the advantage of the proposed approach in that simple keyword filters would have missed more than 50% of the tested attacks, particularly those involving encoding, obfuscation, or newly adopted phrases [4].

Despite strong empirical performance in a small test suite, the current implementation has several limitations. One limitation is the risk score clustering issue that arose when testing the prompts. Since high-confidence pattern detections currently contribute a fixed weight, scores tended to cluster around discrete values such as 5.0, 7.0, and 8.0 rather than forming a smooth distribution of risk scores. This clustering reduces the quality of policy decision and doesn't prioritize alerts for a SOC analyst

as well as it should. Another limitation involves adversarial robustness since the pattern detection engine relies on hand-crafted expressions, which can be evaded through new phrasing, as stated prior. Since the current system is also text-only and does not address multi-modal prompt injection attacks involving images, diagrams, or embedded visual cues, extending the normalization and detection pipeline to handle these cases is necessary [20].

Compared to existing agent security frameworks, the proposed architecture provides broader coverage of modern threat models. NVIDIA NeMo Guardrails is an open-source toolkit for adding programmable guardrails to LLM-based systems [21] and relative to it, both systems support pattern matching and policy enforcement. However, NeMo Guardrails primarily operates post prompt-generation and lacks explicit mechanisms for blocking the prompt from reaching the LLM, contagion detection, and does not have a visualization implementation unlike this system built. When compared to model-level safety approaches such as Anthropic's Constitutional AI as well, the distinction is primarily within the architectural design. Constitutional AI improves model alignment through retraining whereas the proposed system enforces security at runtime [22][23]. Overall, this comparative analysis highlights that while existing defenses address pieces of the problem, none explicitly target multi-agent contagion and execution control in a single architecture.

5 Future Outlook and Research Directions

As AI agents continue to evolve toward greater autonomy, persistence, and collaboration, the security challenges discussed in this work are bound to escalate. In the near term, research efforts should focus on improving how risk is calculated and adversarial robustness to adaptive attacks and novel phrases. If research focuses on combining rule-based detection with learned classifiers using machine learning and deep learning, those two challenges can be refined to better capture threat severity and intent, which is essential for deploying agent security mechanisms at scale.

Over the medium term, securing long-term memory and multi-agent coordination will become increasingly risky. Future systems must treat memory as a security boundary, incorporating validation and periodic sanitization to prevent persistent compromise. Similarly, inter-agent communication should adopt explicit trust models, which can help ensure that outputs from one agent are not blindly consumed by others without verification.

In the long term, as agentic AI systems become more and more integrated in critical infrastructure and business operations, security architectures will need to evolve beyond reactive defenses. Promising research directions include finding behavioral baselines, anomaly detections using temporal analysis, and a formal verification of agent workflows. Ultimately, defending agentic AI will require a shift in mindset: from securing individual prompts or models to securing entire agent ecosystems as time moves forward.

6 Conclusion

This paper examined the emerging security risks posed by agentic AI systems with a specific focus on prompt injection, hybrid attacks, and multi-agent contagion. Through a thorough review of the current state of the art research landscape and existing defenses, this work showed the fundamental gaps in how AI agents are protected today. To address these challenges, the paper introduced a layered AI Agent Middleware Security Layer that integrates content normalization, policy enforcement, and a novel contagion detection layer into a single architecture. Testing of the system showed that this approach can be effective by detecting and blocking a plethora of real-world attacks while maintaining acceptable performance. Most importantly, however, the architecture reframes AI agent security as a system-level issue, rather than an individual or prompt-by-prompt issue. As agentic AI systems continue to escalate across multiple industries, the security of these operations will be a top priority.

References

[1] A. McHugh et al., "Prompt Injection 2.0: Hybrid Attacks Against LLMs," 2024.

[2] L. Liu et al., "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," NeurIPS, 2024.

[3] K. Kraunelis et al., "A Comprehensive Benchmark and Defense Framework for Securing AI Agents Against Prompt Injection," 2024.

[4] S. Gulyamov et al., "Prompt Injection Attacks in LLMs and AI Agent Systems: A Survey," 2024.

[5] F. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques for Language Models," 2022.

[6] Anthropic, "AI Threat Intelligence Update," 2025.

[7] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2025.

[8] T. Tian et al., "A Taxonomy of Prompt Defects in Large Language Model Systems," 2025.

[9] Y. Li et al., "MELON: Provable Defense Against Indirect Prompt Injection Attacks in AI Agents," 2024.

[10] E. Elyoseph et al., "Evaluating Multi-Agent Defenses Against Jailbreaking Attacks on Large Language Models," 2024.

[11] H. Zhang et al., "JailJudge: A Multi-Agent Evaluation Framework for Jailbreak Detection," 2024.

[12] A. McHugh et al., "Breaking the Code: Security Assessment of AI Code Agents Through Systematic Jailbreaking Attacks," 2024.

[13] P. Chhikara et al., "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory," 2025.

[14] I. Ajuzieogu, "Memory Architectures in Long-Term AI Agents," 2025.

[15] S. Willison, "Prompt Injection Explained," 2023.

[16] Y. Liu et al., "Prompt Injection Detection and Mitigation via Multi-Agent NLP Frameworks," 2023.

[17] Z. Chen et al., "Defending Against Prompt Injection with Defensive Tokens," 2025.

[18] A. Piet et al., "Jatmo: Prompt Injection Defense by Task-Specific Fine-Tuning," 2024.

[19] Z. Chen et al., "StruQ: Defending Against Prompt Injection with Structured Queries," 2025.

[20] E. Bagdasaryan et al., "Backdoor Attacks on Vision-Language Models," 2023.

[21] NVIDIA, "NeMo Guardrails: Programmable Safety for LLM Applications," 2024.

[22] A. Bai et al., "Constitutional AI: Harmlessness from AI Feedback," 2022.

[23] OpenAI, "OpenAI Moderation API Documentation," 2024.