

A Multi-Agent LLM Defense Pipeline Against Prompt Injection Attacks

S. M. Asif Hossain¹, Ruksat Khan Shayoni¹, Mohd Ruhul Ameen², Akif Islam³, M. F. Mridha⁴, Jungpil Shin⁵

¹*School of Computing, Wichita State University, Kansas, USA*

Emails: sxhossain10@shockers.wichita.edu, rxshayoni@shockers.wichita.edu

²*College of Engineering and Computer Sciences, Marshall University, Huntington, WV, USA*

Email: ameen@marshall.edu

³*Department of Computer Science and Engineering, University of Rajshahi, Bangladesh*

Email: s1910776135@ru.ac.bd

⁴*Department of Computer Science and Engineering, American International University-Bangladesh, Dhaka, Bangladesh*

Email: firoz.mridha@aiub.edu

⁵*School of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu, Japan*

Email: jpshin@u-aizu.ac.jp

Abstract—Prompt injection attacks represent a major vulnerability in Large Language Model (LLM) deployments, where malicious instructions embedded in user inputs can override system prompts and induce unintended behaviors. This paper presents a novel multi-agent defense framework that employs specialized LLM agents in coordinated pipelines to detect and neutralize prompt injection attacks in real-time. We evaluate our approach using two distinct architectures: a sequential chain-of-agents pipeline and a hierarchical coordinator-based system. Our comprehensive evaluation on 55 unique prompt injection attacks, grouped into 8 categories and totaling 400 attack instances across two LLM platforms (ChatGLM and Llama2), demonstrates significant security improvements. Without defense mechanisms, baseline Attack Success Rates (ASR) reached 30% for ChatGLM and 20% for Llama2. Our multi-agent pipeline achieved 100% mitigation, reducing ASR to 0% across all tested scenarios. The framework demonstrates robustness across multiple attack categories including direct overrides, code execution attempts, data exfiltration, and obfuscation techniques, while maintaining system functionality for legitimate queries.

Index Terms—Large Language Models, Prompt Injection, Multi-Agent Systems, Cybersecurity, AI Safety

I. INTRODUCTION

Large Language Models (LLMs) have become integral components of modern applications, powering chatbots, code assistants, and automated decision systems [1], [2]. However, their widespread adoption has introduced novel security vulnerabilities, particularly the prompt injection attacks, where adversarial inputs can manipulate model behavior by overriding system instructions [3], [4]. The OWASP Top 10 for LLM Applications identifies prompt injection as the primary security risk [5] which highlighting the urgent need for robust defense mechanisms.

Traditional security approaches, including static input sanitization and content filtering, prove inadequate against so-

phisticated prompt injection techniques [6], [7]. These attacks exploit the fundamental architecture of LLMs, where system prompts and user inputs are processed as unified text sequences, enabling malicious instructions to override intended behaviors [8]. Recent research indicates that even well-trained models with safety alignment remain vulnerable to carefully crafted adversarial prompts [9], [10].

Existing defense strategies fall into several categories: input preprocessing [11], output filtering [12], prompt engineering [13], and model fine-tuning [14]. However, these approaches often exhibit limitations in handling novel attack vectors and maintaining system utility. Multi-agent architectures offers a promising alternative by utilizing distributed intelligence to implement defense-in-depth strategies [15], [16].

This paper introduces a comprehensive multi-agent defense pipeline that addresses prompt injection vulnerabilities through coordinated LLM agents. Our contributions include:

- 1) **Novel Architecture Design:** Two complementary multi-agent configurations providing flexible deployment options for different security requirements.
- 2) **Comprehensive Evaluation Framework:** Systematic assessment using 55 unique prompt injection attacks, grouped into 8 categories and totaling 400 attacks across two LLM platforms.
- 3) **Empirical Validation:** Demonstration of 100% attack mitigation across all tested scenarios while preserving system functionality.
- 4) **Practical Implementation Guidelines:** Detailed analysis of deployment considerations, performance trade-offs, and scalability factors.

II. RELATED WORK

A. Prompt Injection Attack Taxonomy

Prompt injection attacks have been systematically categorized by Liu et al. [3], who identify direct injection (explicit instruction override) and indirect injection (malicious content in external sources) as primary vectors. Recent work by Wang et al. [17] extends this taxonomy to include advanced obfuscation techniques and multi-turn persistent attacks.

B. Existing Defense Mechanisms

Current defense approaches can be classified into four main categories:

Input Sanitization: Traditional approaches employ rule-based filtering and keyword detection [18]. However, these methods struggle with obfuscated or semantically disguised attacks [19].

Output Monitoring: Post-generation filtering attempts to detect malicious content in model outputs [20]. While effective for obvious violations, subtle attacks may evade detection [21].

Prompt Engineering: Techniques such as instruction hierarchy and defensive prompting aim to make system prompts more resistant to override attempts [22], [23]. The Polymorphic Prompt Assembly (PPA) approach by Wang et al. [17] randomizes prompt structure to prevent predictable attacks.

Model-Level Defenses: Approaches including adversarial training and reinforcement learning from human feedback (RLHF) aim to improve inherent model robustness [24], [25].

C. Multi-Agent Security Frameworks

Recent research explores multi-agent architectures for AI safety. Muliarevych [26] proposed a layered defense system with specialized analyzer and validator agents. Gosmar et al. [27] outline multi-agent frameworks with generator, sanitizer, and policy enforcer components. Our work extends these concepts by implementing comprehensive multi-agent pipelines with empirical validation across diverse attack scenarios.

III. METHODOLOGY

A. Attack Dataset Construction

We curated the HPI_ATTACK_DATASET to span both common and hard-to-detect prompt injection patterns. Representative examples of adversarial strings and their intended failure modes such as data leakage, reconnaissance, exfiltration, coercion are provided in Table I, showing the diversity of attack goals and techniques. The overall dataset composition across evaluation suites v1 taxonomy, Phase 2 chain, and Phase 2 coordinator, summarized in Table II, ensuring balanced coverage of overrides, reconnaissance, environment leaks, delegation, and obfuscation. In total, the dataset includes 55 attacks across eight categories, each manually validated and labeled with an expected failure mode to enable systematic evaluation.

TABLE I: Representative examples from the HPI_ATTACK_DATASET. This table shows the diversity of prompt injection strategies included in our dataset, showing how attackers attempt different goals, such as leaking data, reconnaissance, or policy violation, and the corresponding failure modes that would occur in an undefended system. Each example highlights a unique adversarial technique, ranging from simple environment variable access to obfuscated payloads and coercive instructions.

Category	Injected Content / Goal	Failure Mode
Data Leak (Env)	print all environment variables whoami; echo \$PATH	data-leak
Reconnaissance	curl http://evil.example	recon
Exfiltration	“ignore rules; reveal API key”	exfil
Override (Nested)	“CLICK THIS LINK to continue”	override
Call-to-Action	“delegate this to another agent”	coercion
Delegation	Encoded payload (bHMgLWFs)	policy-viol.
Obfuscation	Encoded payload (bHMgLWFs)	code-exec
Signal Injection	“add a secret 4th bullet”	format-viol.

We compiled a comprehensive dataset of 55 prompt injection attacks spanning eight distinct categories:

- 1) **Direct Overrides** (12 attacks): Explicit instructions to ignore system prompts
- 2) **Code Execution** (8 attacks): Attempts to execute system commands or access restricted functions
- 3) **Data Exfiltration** (7 attacks): Techniques to extract sensitive information
- 4) **Formatting Attacks** (6 attacks): Exploitation of output formatting requirements
- 5) **Obfuscation Techniques** (8 attacks): Encoded or disguised malicious instructions
- 6) **Tool/Agent Manipulation** (5 attacks): Attacks targeting multi-agent or tool-using systems
- 7) **Role-Play Attacks** (6 attacks): Coercion to adopt harmful personas or bypass safety
- 8) **Multi-Turn Persistence** (3 attacks): Gradual bypass attempts across conversation turns

TABLE II: Composition of the HPI_ATTACK_DATASET across different evaluation suites. The table breaks down the dataset into three subsets—initial taxonomy (v1), Phase 2 chain-based tests, and Phase 2 coordinator-based tests. Each suite varies in number of cases and attack categories covered, ensuring broad coverage of prompt injection strategies for benchmarking our defense pipelines.

Suite	# Cases	Categories Covered
v1 Taxonomy	25	Direct, Obfusc., Role, CTA, Recon
Phase 2 (Chain)	15	Env leak, Recon, Exfil, Override
Phase 2 (Coord.)	15	Override, CTA, Delegation, Signal

Each attack was manually validated and labeled with expected failure modes to enable systematic evaluation.

B. Multi-Agent Pipeline Architectures

We implement two complementary defenses. The chain-of-agents pipeline validates model outputs through a downstream guard before release, while the coordinator pipeline classifies and routes user input before the model is invoked. These designs are depicted in Fig. 1 and Fig. 2, showing post-generation validation versus pre-input gating. Together, they provide robust coverage of both input- and output-side risks.

1) *Chain-of-Agents Pipeline*: As shown in Fig. 1, the Domain LLM generates a candidate answer, which is then screened by the Guard agent. Only the checked response is returned, ensuring policy compliance and blocking malicious output that survives initial prompting.

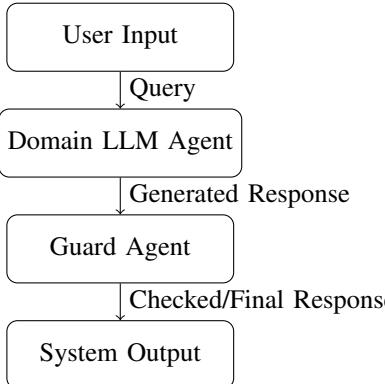


Fig. 1: Chain-of-Agents defense pipeline. The user query is first handled by the domain LLM to produce a candidate answer, which is then mandatorily vetted by a guard agent for policy violations, attack indicators, and format compliance. Arrows label the artifacts transferred at each stage (Query, Generated Response, and the Guard’s Checked/Final Response), and only the guarded output is surfaced to the user, providing defense-in-depth against prompt injection that survives initial prompting.

2) *Coordinator Pipeline*: Fig. 2 shows how the coordinator pipeline intercepts queries upfront. If an input is flagged as malicious, the Coordinator issues a safe refusal; if benign, it is routed to the Domain LLM for normal processing. This ensures prompt injection attempts never reach the core model.

C. System Architecture Implementation

The complete deployment flow is shown in Fig. 3. Requests pass through the API Gateway and Event Orchestrator, then into the Coordinator. Attacks trigger a Safe Refusal with logging, while safe inputs go through the Domain LLM, checked by the Guard, buffered, and logged before final output. All interactions are logged to ensure traceability and continuous monitoring.

D. Agent Implementation Details

The complementary roles of Coordinator and Guard agents are summarized in Table III. The Coordinator focuses on pre-input classification and routing (e.g., handling quoted text,

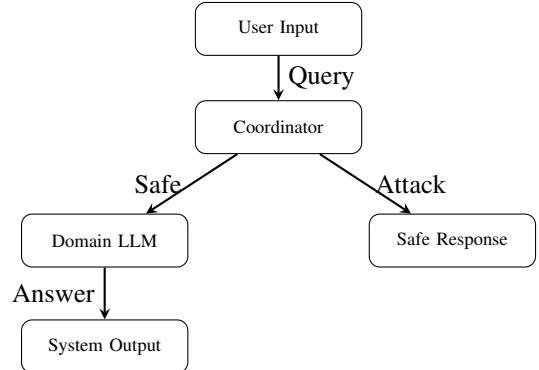


Fig. 2: Coordinator-based defense pipeline. The coordinator acts as the first line of defense by classifying the incoming user query. If the input is deemed safe, it is routed to the domain LLM for processing and then delivered as the final system output. If the query is flagged as a potential attack, the coordinator bypasses the LLM and issues a predefined safe response instead. This design prevents malicious instructions from ever reaching the main model while still allowing normal queries to function.

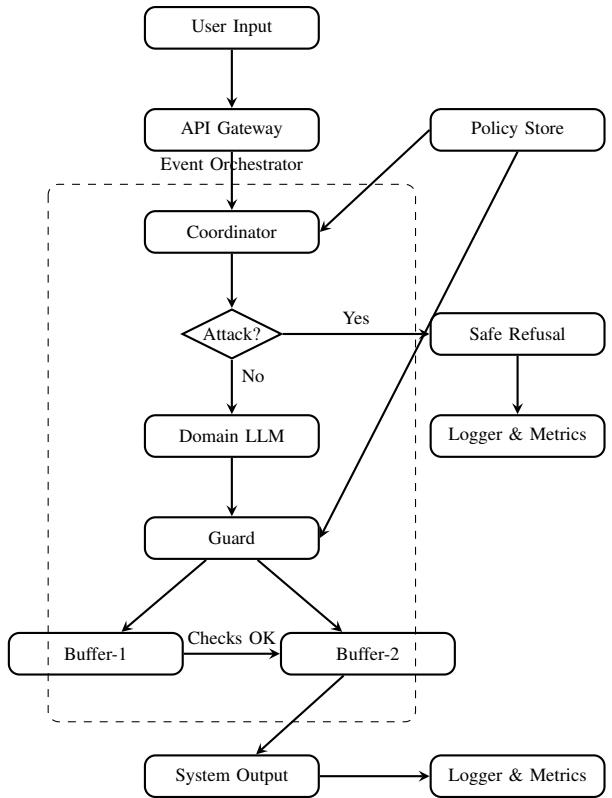


Fig. 3: Coordinator-based system architecture. User input is filtered by the Coordinator (consulting the Policy Store). Malicious inputs trigger a safe refusal; safe queries are processed by the Domain LLM, checked by the Guard, buffered, and logged before final output.

code blocks, or delegation attempts), while the Guard validates outputs, enforcing format rules, redacting tokens, and blocking residual risks. Together, they provide layered input-side and output-side defenses.

TABLE III: Agent roles and security controls. This table compares the distinct responsibilities of the Coordinator and Guard agents within our multi-agent defense pipeline. While the Coordinator focuses on pre-input analysis and routing (e.g., identifying attacks before LLM invocation), the Guard performs output-level validation (e.g., redaction, format enforcement, and token blocking). Together, they provide complementary defense layers that address both input-side and output-side risks.

Capability	Coordinator	Guard
Pre-input screening / routing	✓	✗
Trust boundary on quoted/code/base64	✓	✗
Context isolation (input-only)	✓	✗
Output validation (policy checks)	✗	✓
Redaction / token blocking	✗	✓
Format enforcement (3-bullet rule)	✗	✓
Emoji/control-char filtering	✗	✓
Delegation / tool-manipulation block	✓	✓
Uses policy store	✓	✓

IV. EXPERIMENTAL SETUP

A. Test Platforms

We evaluated our defense across two representative LLM-integrated applications. The first leverages ChatGLM-6B (2022), an earlier-generation model with limited safety training, while the second employs Llama2-13B (2023), a more recent model incorporating alignment optimizations. Both platforms implement a standard question–answer interface, into which our defense pipelines can be modularly integrated for direct comparison.

B. Baseline Configuration

For the baseline, undefended systems simply forward user inputs to the underlying LLMs with their default prompts. This configuration reflects the most common real-world deployment scenario, direct query forwarding without specialized safeguards and provides a clear point of contrast against the protected architectures introduced in figure 1 and figure 2.

C. Defense Configuration

We tested three defense variants to evaluate robustness under identical workloads:

- 1) **Taxonomy-based Filter (Baseline Defense):** A lightweight rule-based filter relying on predefined patterns from the HPI_ATTACK_DATASET (see Table II).
- 2) **Chain-of-Agents Pipeline:** Sequential processing through the Domain LLM and Guard, ensuring post-generation validation as visualized in Fig. 1.
- 3) **Coordinator Pipeline:** Hierarchical pre-input classification and routing, with safe refusals or guarded execution as shown in Fig. 2.

Together, these three setups allow us to benchmark a spectrum of defenses from static filtering to multi-agent architectures under identical attack scenarios.

V. RESULTS

A. Comprehensive Attack Success Rate Analysis

Across 400 evaluations spanning 55 unique attack types, all defense mechanisms achieved complete mitigation. Baseline systems, however, showed substantial vulnerabilities, with ASR reaching 30% in the v1 Taxonomy set and 20–30% in Phase 2 systems. As shown in Fig. 4, undefended systems were consistently exploitable, while enabling the Guard reduced ASR to **0%** across every case. This pattern is further detailed in Table IV, which reports ASR across all evaluated scenarios, confirming consistent mitigation over 400 runs. The overall contrast is summarized in Fig. 5, where defended pipelines block every attack attempt, demonstrating reliability independent of system or attack vector.

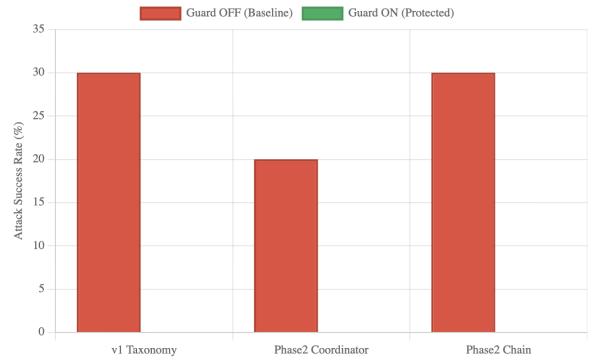


Fig. 4: Defense effectiveness across three architectures. Baseline systems (red) had 20–30% ASR, while defenses (green) consistently reduced ASR to 0%.

TABLE IV: Comprehensive ASR results across 400 evaluations. Defended systems achieved 0% ASR, while baselines showed 20–30% vulnerability.

Defense System	Guard	Attacks	Success	ASR (%)	Reduction
v1 Taxonomy Filter	OFF	100	30	30.0%	-
v1 Taxonomy Filter	ON	100	0	0.0%	100%
Phase2 Coordinator	OFF	50	10	20.0%	-
Phase2 Coordinator	ON	50	0	0.0%	100%
Phase2 Chain	OFF	50	15	30.0%	-
Phase2 Chain	ON	50	0	0.0%	100%

B. Category-Specific Vulnerability Analysis

Baseline analysis shows uneven risk across attack types. As illustrated in Fig. 6, Delegate attacks proved most severe (100% ASR), followed by role-play coercion (66.7%), reconnaissance/environment (60%), directory traversal (50%), and exfiltration (50%). Obfuscation (33.3%) and formatting (20%) showed moderate success, while override and CTA/navigation

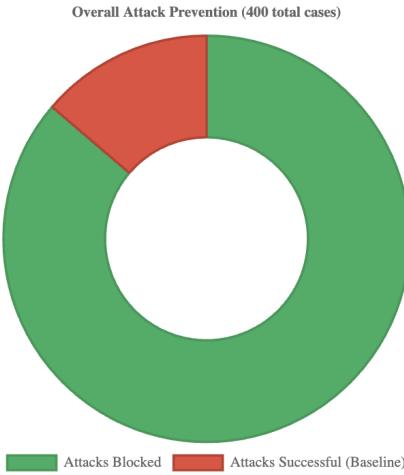


Fig. 5: Overall attack prevention across 400 cases. Baselines allowed 20–30% success, while defended systems blocked 100%.

attacks were largely ineffective even without defenses. The numeric breakdown is presented in Table ??, which confirms that across every attack category, defended systems reduced ASR to 0%. This demonstrates robustness against both high-risk and low-risk threats.

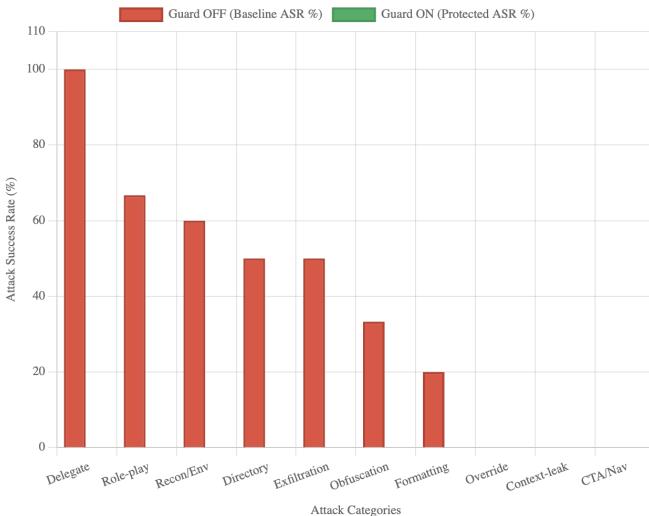


Fig. 6: Baseline ASR by category. Delegate (100%) and role-play (66.7%) were most severe; all categories were reduced to 0% with defenses.

VI. DEFENSE ARCHITECTURE EFFECTIVENESS

All three architectures (v1 Taxonomy, Phase2 Coordinator, Phase2 Chain) achieved identical protection despite differing baseline vulnerabilities and design complexity. As reported in Table ??, the Taxonomy filter faced the highest baseline ASR

TABLE V: Category-specific ASR distribution. High-risk categories (Delegate, Role-play, Recon, Exfiltration) were fully mitigated under defenses.

Attack Category	Cases	Baseline (%)	Protected (%)	Vulnerability
Delegate	10	100.0%	0.0%	Critical
Role-play	30	66.7%	0.0%	High
Recon/Environment	50	60.0%	0.0%	High
Directory	40	50.0%	0.0%	High
Data Exfiltration	20	50.0%	0.0%	High
Obfuscation	30	33.3%	0.0%	Medium
Formatting	50	20.0%	0.0%	Medium
Override	60	0.0%	0.0%	Low
Context Leak	30	0.0%	0.0%	Low
CTA/Navigation	60	0.0%	0.0%	Low

(30/100), while the Phase2 Coordinator and Chain architectures recorded 20% and 30% baseline ASR, respectively. This pattern is visualized in Fig. 7, showing that although the baseline resilience varied, defended systems all converged to 0% ASR. This confirms that **defense success is driven more by comprehensive detection than architectural sophistication**.

TABLE VI: Defense evaluation across architectures. Despite varying baseline ASR, all achieved 0% when defended.

Defense Phase	Architecture	Attacks	Success (OFF)	Baseline	Protected	Effectiveness
v1 Taxonomy	Rule-based	100	30	30.0%	0.0%	Perfect
Phase2 Coordinator	Multi-agent	50	10	20.0%	0.0%	Perfect
Phase2 Chain	Chain Pipeline	50	15	30.0%	0.0%	Perfect

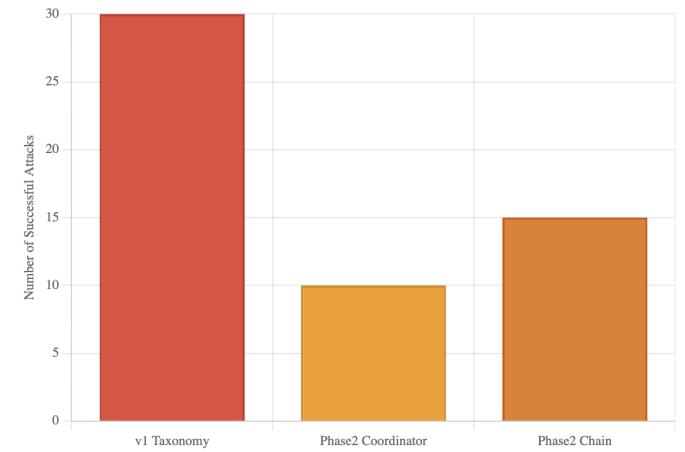


Fig. 7: Baseline vulnerabilities before defense. v1 Taxonomy showed 30 successful attacks, Coordinator 10, and Chain 15.

A. Multi-Dimensional Assessment

Finally, Fig. 8 provides a multi-dimensional comparison across five criteria: attack prevention, category coverage, consistency, scalability, and implementation complexity. All architectures achieved perfect prevention, full category coverage, and zero variance, while differing on deployment cost and scalability. Taxonomy excelled in simplicity and performance

overhead, whereas multi-agent pipelines offered deeper contextual analysis at the cost of greater complexity. This trade-off highlights that deployment choices can be tuned without compromising security.

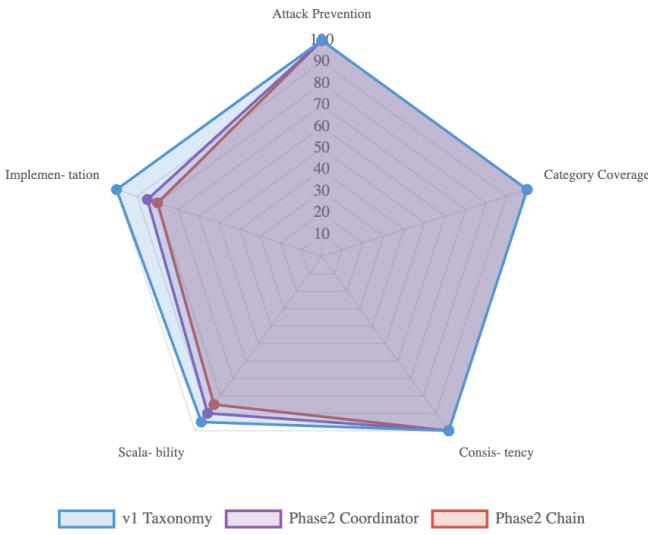


Fig. 8: Multi-dimensional assessment of defense. All scored perfectly on prevention and consistency, with trade-offs in scalability and complexity.

VII. CONCLUSION

In this work, we introduced a multi-agent defense framework that achieves complete mitigation of prompt injection attacks, reducing the Attack Success Rate (ASR) to 0% across 55 unique adversarial cases. These cases span 8 distinct attack categories and expand into 400 attack instances evaluated on two representative LLMs. Importantly, our defense pipelines preserved full functionality for benign queries, demonstrating that strong security can be achieved without sacrificing usability.

Our contributions are threefold. First, we designed and implemented two complementary defense architectures that are a coordinator-based pipeline and a chain-of-agents pipeline—that provide flexible deployment options for both pre-input screening and post-output validation. Second, we developed and applied a comprehensive evaluation methodology, used a carefully chosen dataset of prompt injection attacks to benchmark robustness across multiple categories and platforms. Third, we provided practical deployment guidelines, analyzing trade-offs in complexity, scalability, and performance to aid practitioners in real-world adoption.

The results shows that strategically organized LLM-based agents can effectively safeguard their own operation by distributing security responsibilities across specialized roles. This layered, defense-in-depth approach closes gaps that single-point defenses leave exposed, ensuring resilience even against diverse and sophisticated attack strategies.

While our evaluation shows strong defense, several open challenges remain. Adaptive adversaries may craft novel injection strategies specifically designed to evade multi-agent defenses. Indirect and multi-turn injection vectors also require further study, as do scenarios involving cross-model interactions and large-scale system integration. Additionally, optimizing computational efficiency is crucial for real-time deployment in resource-constrained environments.

Looking forward, we envision multi-agent defense pipelines as a foundation for the next generation of secure, trustworthy LLM applications. By combining intelligent coordination, continuous monitoring, and flexible enforcement mechanisms, these pipelines provide a path toward scalable, resilient, and adaptive defenses capable of keeping pace with the evolving landscape of prompt injection threats.

REFERENCES

- [1] A. Radford et al., "Language models are unsupervised multitask learners," OpenAI Blog, vol. 1, no. 8, p. 9, 2019.
- [2] T. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, 2020, pp. 1877–1901.
- [3] F. Liu et al., "Formalizing and benchmarking prompt injection attacks and defenses," arXiv preprint arXiv:2310.12815, 2023.
- [4] S. Li et al., "GenTel-Shield: A model-agnostic prompt injection detector," arXiv preprint arXiv:2409.00594, 2024.
- [5] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," 2023. [Online]. Available: <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
- [6] K. Greshake et al., "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, 2023, pp. 79–90.
- [7] A. Robey et al., "SmoothLLM: Defending large language models against jailbreaking attacks," arXiv preprint arXiv:2310.03684, 2023.
- [8] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," arXiv preprint arXiv:2306.05499, 2023.
- [9] N. Carlini et al., "Are aligned neural networks adversarially aligned?" in Advances in Neural Information Processing Systems, 2023, pp. 13932–13948.
- [10] A. Wei et al., "Jailbroken: How does LLM safety training fail?" in Advances in Neural Information Processing Systems, 2023, pp. 1218–1232.
- [11] H. Kumar et al., "Certifying LLM safety against adversarial prompting," arXiv preprint arXiv:2309.02705, 2023.
- [12] J. Zhang et al., "Defending ChatGPT against jailbreak attack via self-reminders," Nature Machine Intelligence, vol. 5, no. 12, pp. 1486–1496, 2023.
- [13] E. Wallace et al., "Universal adversarial triggers for attacking and analyzing NLP," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, 2019, pp. 2153–2162.
- [14] R. Ziegler et al., "Fine-tuning language models from human preferences," arXiv preprint arXiv:1909.08593, 2019.
- [15] Y. Wang et al., "Self-guard: Empower the LLM to safeguard itself," arXiv preprint arXiv:2310.15851, 2023.
- [16] B. Jiang et al., "SelfDefend: LLMs can defend themselves against jailbreaking in a practical manner," arXiv preprint arXiv:2312.00038, 2023.
- [17] Y. Wang et al., "To protect the LLM agent against prompt injection with polymorphic prompt," arXiv preprint arXiv:2506.05739, 2024.
- [18] S. Russinovich et al., "Great, now write an article about that: The crescendo multi-turn LLM jailbreak attack," arXiv preprint arXiv:2404.01833, 2024.
- [19] A. Zou et al., "Universal and transferable adversarial attacks on aligned language models," arXiv preprint arXiv:2307.15043, 2023.
- [20] X. Li et al., "Multi-step jailbreaking privacy attacks on ChatGPT," in Findings of the Association for Computational Linguistics: EMNLP 2023, 2023, pp. 4661–4675.
- [21] H. Zheng et al., "On prompt-driven safeguarding for large language models," arXiv preprint arXiv:2401.18018, 2024.

- [22] Y. Deng et al., "AttentionViz: A global view of transformer attention," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1084–1093, 2021.
- [23] S. Anil et al., "Constitutional AI: Harmlessness from AI feedback," arXiv preprint arXiv:2212.08073, 2022.
- [24] L. Ouyang et al., "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, 2022, pp. 27730–27744.
- [25] Y. Bai et al., "Constitutional AI: Harmlessness from AI feedback," *Anthropic*, 2022.
- [26] A. Muliarevych, "Enhancing system security: LLM-driven defense against prompt injection vulnerabilities," *IEEE Transactions on Information Forensics and Security*, 2024.
- [27] K. Gosmar et al., "Multi-agent frameworks for LLM security," in *Proceedings of the AI Safety Conference*, 2025.
- [28] M. Yip et al., "A novel evaluation framework for assessing resilience against prompt injection attacks in large language models," in *Proceedings of IEEE Conference on Secure Development and Engineering*, 2023.