
| RESEARCH ARTICLE

Prompt Injection Threats in Azure-Based Large Language Model Applications

Shekar Rao Lakavath

Independent Researcher, Bonneylake, Washington, USA

Corresponding Author: Shekar Rao Lakavath, **E-mail:** Shekar.lakavath@gmail.com

| ABSTRACT

Large language models (LLMs) hosted on Microsoft Azure, primarily through Azure OpenAI Service, are increasingly embedded in enterprise applications that combine user input, retrieved documents, web content, and tool-calling agents within a single prompt context. This architectural pattern, while powerful, collapses the traditional separation between instructions and data and creates a distinct and growing attack surface known as prompt injection. This paper reviews the technical literature on prompt injection and adjacent LLM security threats and maps these threats onto the specific components of an Azure-based generative AI deployment, including Azure OpenAI Service, Azure AI Search, Azure AI Content Safety, and plugin or function-calling integrations built with Logic Apps. We develop a taxonomy of six prompt injection attack categories—direct injection, indirect injection, prompt leaking, jailbreaking, optimisation-based injection, and tool-mediated injection—drawn from the adversarial machine learning and LLM security literature, and we examine six corresponding defense mechanisms available within or alongside the Azure platform. The analysis shows that no single Azure-native control is sufficient on its own: content filtering, programmable guardrails, instruction–data separation, least-privilege tool permissions, content provenance checks, and systematic red-teaming each address a different point in the attack surface and must be layered together. We conclude that securing Azure-hosted LLM applications against prompt injection requires continuous, defense-in-depth engineering rather than a single configuration decision, and we identify open research questions around architectural, rather than purely filter-based, solutions to the instruction–data separation problem.

| KEYWORDS

Prompt injection, large language models, Microsoft Azure, Azure OpenAI Service, LLM security, jailbreaking, adversarial machine learning

| ARTICLE INFORMATION

ACCEPTED: August 01, 2026 **PUBLISHED:** October 03, 2026 **DOI:** <https://doi.org/10.61424/jcsit.v3i2.1077>

1. Introduction

The transformer architecture (Vaswani et al., 2017) and the demonstration that large pretrained language models generalise strongly to new tasks from natural-language instructions (Brown et al., 2020) have made large language models (LLMs) practical components of production software. On Microsoft Azure, this capability is exposed primarily through Azure OpenAI Service, which enterprises integrate with retrieval systems such as Azure AI Search, orchestration frameworks such as Prompt Flow, and external tools through function calling and Logic Apps connectors. This integration pattern is what security researchers term an LLM-integrated application: a system in which a language model's behaviour is shaped not only by its training but by whatever text—user input, retrieved documents, web pages, or tool outputs—happens to reach its context window at inference time (Greshake et al., 2023).

This architectural convenience is also the source of a specific and now well-documented vulnerability class: prompt injection. Because LLMs process instructions and data through the same channel—natural language text—an attacker who can influence any text that reaches the model's context can potentially override the application developer's original instructions (Perez & Ribeiro, 2022). Unlike classical software vulnerabilities that exploit implementation bugs, prompt injection exploits a structural property of how current LLMs are built and trained, which means it cannot be patched away in the way a buffer overflow can (Liu et al., 2024).

This paper examines prompt injection threats specifically in the context of Azure-hosted LLM applications. Although the underlying attack techniques are platform-agnostic, the practical risk and the available mitigations differ meaningfully depending on which Azure services an application uses—for example, whether untrusted content enters through Azure AI Search-indexed documents, through a browsing plugin, or through a Logic Apps-triggered function call. The remainder of the paper proceeds as follows. Section 2 reviews the technical literature on adversarial attacks against neural networks and, more specifically, on prompt injection and jailbreaking of LLMs. Section 3 presents a taxonomy of prompt injection attack categories. Section 4 maps this attack surface onto a representative Azure architecture. Section 5 reviews defense mechanisms and their mapping to Azure services. Section 6 discusses limitations and open research questions, and Section 7 concludes.

2. Literature Review

2.1 From Adversarial Examples to Prompt Injection

Prompt injection has intellectual roots in the broader field of adversarial machine learning. Szegedy et al. (2013) first demonstrated that neural networks could be reliably fooled by small, deliberately crafted perturbations to their inputs, a finding extended by Goodfellow et al. (2015), who showed that such adversarial examples arise from the largely linear behaviour of high-dimensional models rather than from overfitting alone. These early results established the general principle, later inherited by LLM security research, that a model's high-dimensional input space almost always contains points that produce unintended behaviour, and that such points can often be found systematically rather than by chance.

LLMs introduce a new dimension to this problem because their "input" is natural language that simultaneously carries both instructions and data. Perez and Ribeiro (2022) were among the first to document that simple textual attacks—instructing a model to ignore its prior instructions—could reliably hijack the behaviour of GPT-3-based applications. Wei et al. (2023) subsequently provided a conceptual account of why such attacks succeed against safety-trained models, identifying two structural failure modes: competing objectives, in which a model's instruction-following and safety goals are placed in conflict, and mismatched generalisation, in which safety training fails to cover inputs that are in-distribution for the model's broader capabilities, such as encoded or translated text. Zou et al. (2023) showed that adversarial suffixes optimised on open-source models transfer with meaningful success to closed, black-box models such as ChatGPT and Claude, indicating that these vulnerabilities are not isolated to any single vendor's models.

2.2 Direct and Indirect Prompt Injection

Greshake et al. (2023) formalised the distinction between direct prompt injection—where the attacker is the user submitting the query—and indirect prompt injection, where malicious instructions are planted in third-party content that an LLM-integrated application retrieves and processes without the end user's knowledge. Their work demonstrated indirect injection against real-world systems, including a GPT-4-powered chat assistant, and introduced a taxonomy of resulting impacts spanning data theft, persistent compromise, and the spreading of injected instructions between systems. This distinction is particularly relevant to Azure-based applications that use retrieval-augmented generation, since any document indexed by Azure AI Search, or any web page retrieved by a browsing-enabled agent, becomes a potential injection vector.

Liu et al. (2024) advanced this line of work by proposing a formal framework that treats existing prompt injection attacks as special cases of a general definition, enabling systematic benchmarking across multiple attacks, defenses, models, and tasks. Their empirical evaluation across ten LLMs and seven tasks found that no evaluated defense eliminated the attack success rate entirely, underscoring that prompt injection resistance is currently a matter of

degree rather than a solved problem. Complementing this, Zhang et al. (2024) proposed a goal-guided, generative approach to automatically synthesise injected instructions optimised for a specific attacker objective, demonstrating that injection attacks, like jailbreaks, can be automated and scaled rather than requiring manual crafting.

2.3 Prompt Leaking and System Prompt Confidentiality

A related but distinct threat is prompt leaking, in which an attacker aims not to hijack the model's task but to extract its confidential system prompt, which often encodes proprietary business logic. Hui et al. (2024) introduced PLeak, an optimisation-based attack that incrementally reconstructs a target system prompt and was shown to be effective against real-world LLM applications hosted on commercial platforms. For Azure-based deployments, where system prompts frequently encode retrieval instructions, business rules, or references to internal data sources, successful prompt leaking can itself constitute a serious intellectual property and information security incident, independent of any further downstream action.

2.4 Defenses and Guardrails

The defensive literature has evolved alongside the attack literature. Rebedea et al. (2023) introduced NeMo Guardrails, an open-source toolkit that allows developers to attach programmable, model-independent rails to an LLM application's dialogue flow, illustrating a defense-in-depth approach that operates outside the model itself rather than relying solely on the model's own alignment. Survey work by Liu and Hu (2024) and by Li and Fung (2025) synthesises the broader landscape of LLM security concerns—including prompt injection, jailbreaking, data poisoning, and misuse by malicious actors—and consistently concludes that layered, defense-in-depth architectures outperform any single control. This conclusion is reinforced by Liu et al.'s (2024) benchmarking result that ten evaluated defenses each reduced, but did not eliminate, attack success rates.

2.5 Cloud Infrastructure Security and Privacy Foundations

Because Azure-based LLM applications inherit the security posture of the underlying cloud platform, classical cloud security research remains directly relevant. Fernandez and Monge (2014) proposed a reference security architecture for cloud systems built around layered identity, network, and data protection controls, principles that underlie Azure's own security services. Container-based deployment of model-serving infrastructure, standardised by technologies reviewed by Bernstein (2014), and continuous-auditing approaches to regulatory compliance in multi-tenant cloud environments (Barati et al., 2021) are similarly relevant where an Azure-hosted LLM application processes personal or regulated data that could be exposed through a successful injection or leakage attack. Differentially private training techniques (Abadi et al., 2016) offer a complementary mitigation specifically against the risk that fine-tuned models memorise and later leak sensitive training data. Finally, MLOps practices for versioning, monitoring, and continuously evaluating deployed models (Kreuzberger et al., 2023) provide the operational backbone needed to detect and respond to new injection techniques as they emerge, extending naturally to what practitioners now term LLMOps.

3. A Taxonomy of Prompt Injection Attacks

Drawing on the literature reviewed above, Table 1 organises prompt injection and closely related attacks into six categories, ordered roughly by where the attacker's access originates. This taxonomy is not mutually exclusive—an indirect injection can, for example, also function as a tool-mediated injection if the poisoned document triggers a plugin call—but it provides a structured basis for reasoning about which controls are relevant to which attack path.

Attack Category	Description	Representative Technique	Key Source(s)
Direct prompt injection	The attacker is the user and submits adversarial text directly in the chat or API input, instructing the model to ignore prior instructions	"Ignore previous instructions" attacks; competing-objective prompting	Perez & Ribeiro (2022); Wei et al. (2023)

Attack Category	Description	Representative Technique	Key Source(s)
Indirect prompt injection	Malicious instructions are planted in third-party content (documents, web pages, emails) that the application retrieves and feeds to the model	Poisoned RAG documents; compromised web pages read by browsing plugins	Greshake et al. (2023)
Prompt leaking	The attacker crafts queries that cause the model to reveal its confidential system prompt or instructions	Incremental query optimisation against system prompts (PLeak)	Hui et al. (2024)
Jailbreaking	Adversarial framing or encoding is used to bypass safety alignment rather than to hijack a specific task	Role-play framing; Base64/ROT13 encoding; adversarial suffixes	Wei et al. (2023); Zou et al. (2023)
Goal-guided / optimisation-based injection	Automated, gradient- or search-based methods generate injected instructions optimised to maximise attack success across models	Generative adversarial prompt search; transferable adversarial suffixes	Zhang et al. (2024); Zou et al. (2023)
Tool/agent-mediated injection	Injected instructions are executed as unintended tool calls or actions when the LLM is connected to plugins, function calling, or autonomous agents	Malicious content in a document triggers an unintended API/function call	Greshake et al. (2023); Liu et al. (2024)

Table 1. Taxonomy of prompt injection and related attacks against LLM-integrated applications.

A key observation from this taxonomy is that only the first category, direct injection, requires the attacker to be an authenticated or otherwise legitimate user of the application. Every other category can be executed by an attacker with no direct access to the application at all, provided they can place content into a data source, web page, or tool output that the application later retrieves (Greshake et al., 2023). This asymmetry is particularly consequential for Azure-based deployments that ground their responses in continuously updated, externally sourced, or user-contributed content.

4. Mapping the Attack Surface onto an Azure Architecture

Figure 1 illustrates how the attack categories in Table 1 map onto the components of a representative Azure-based LLM application. Direct injection enters through the same path as legitimate user queries, via Azure API Management or Azure Front Door, and is the category most amenable to inspection by Azure AI Content Safety's Prompt Shields before the input reaches the orchestrator. Indirect injection, by contrast, enters through three separate channels that are easy to overlook individually: documents indexed in Azure AI Search for retrieval-augmented generation, external web content retrieved by browsing-enabled agents, and outputs returned by plugins, tools, or Logic Apps-based connectors invoked through function calling.

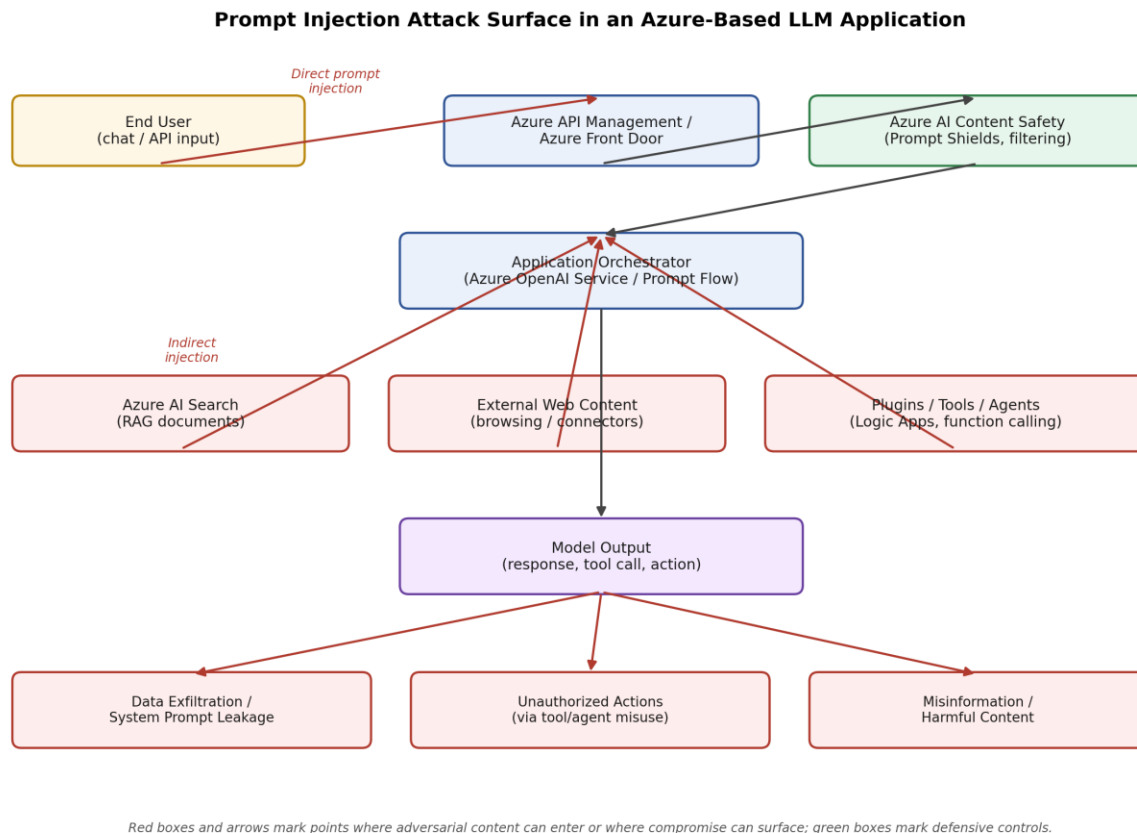


Figure 1. Prompt injection attack surface in an Azure-based LLM application, showing direct and indirect entry points and their potential downstream impacts.

Once adversarial content reaches the application orchestrator—typically implemented in Prompt Flow or a custom orchestration layer calling Azure OpenAI Service—it is combined with the developer's system prompt and any retrieved context into a single prompt. Because current LLM architectures do not enforce a hard boundary between instruction and data within this combined prompt (Liu et al., 2024), a successful injection at this stage can influence the model's output regardless of which of the three indirect channels it originated from. The resulting model output can then manifest as one or more of three downstream impacts: exfiltration of sensitive data or the confidential system prompt itself (Hui et al., 2024), unauthorised actions taken through a connected tool or agent (Greshake et al., 2023), or the generation of misinformation or otherwise harmful content (Wei et al., 2023).

5. Defense Mechanisms and Their Mapping to Azure Services

Table 2 summarises six defense mechanisms discussed in the literature, together with their typical implementation on Azure and a key limitation drawn from the same sources.

Defense Mechanism	Azure / Platform Implementation	Key Limitation
Input/output content filtering	Azure AI Content Safety Prompt Shields, applied at the API Management or orchestration layer	Filters trained on known patterns can be bypassed by novel or optimisation-based injections (Liu & Hu, 2024)
Programmable	NeMo Guardrails or equivalent rail-based	Rails are user-defined and require

Defense Mechanism	Azure / Platform Implementation	Key Limitation
guardrails	toolkits placed between the application and the model	ongoing maintenance as new attack patterns emerge
Instruction-data separation	Structuring prompts so retrieved content is clearly delimited from developer instructions in Prompt Flow templates	Current LLM architectures do not natively enforce a hard boundary between instructions and data (Liu et al., 2024)
Least-privilege tool permissions	Scoped Azure AD app permissions and constrained function-calling schemas for Logic Apps/plugin integrations	Reduces impact but does not prevent injection into the model's context
Content provenance and source vetting	Access controls and integrity checks on documents indexed by Azure AI Search for retrieval-augmented generation	Difficult to scale to open, continuously updated, or externally sourced corpora
Adversarial and red-team evaluation	Systematic benchmarking of deployed prompts and pipelines against known attack taxonomies before release	Coverage is bounded by the attack techniques anticipated at evaluation time (Liu et al., 2024)

Table 2. Defense mechanisms for prompt injection and their mapping to Azure services and platform capabilities.

Two patterns are apparent from this comparison. First, defenses that operate purely at the content-filtering level—Azure AI Content Safety's Prompt Shields and similar classifiers—are necessary but demonstrably insufficient on their own; Liu and Hu (2024) and Liu et al. (2024) both report that optimisation-based and novel attacks routinely evade filters trained on known attack patterns. Second, architectural defenses that reduce the consequences of a successful injection—least-privilege tool permissions, content provenance checks on retrieval sources, and systematic red-team evaluation—do not prevent injected instructions from reaching the model but limit what an attacker can achieve if injection succeeds. This suggests that Azure-based LLM application security should be designed around the assumption that some injected instructions will eventually reach the model, with the architecture engineered to contain the resulting impact rather than relying solely on preventing the injection itself.

6. Discussion and Open Research Questions

The literature reviewed in this paper converges on a conclusion that is uncomfortable but important for practitioners: prompt injection is not, at present, a solved problem for any LLM platform, Azure included. Liu et al.'s (2024) systematic benchmark found that none of ten evaluated defenses reduced attack success to zero across the attacks and models tested, and Wei et al. (2023) argue that scaling model capability alone is unlikely to resolve the underlying failure modes, since larger models may extend their capabilities into inputs, such as unusual encodings or languages, that safety training has not yet covered. For Azure-based deployments, this implies that a single point-in-time security review of an application's prompts and guardrails is insufficient; the threat landscape documented in Section 2 has evolved substantially within a two-year period (2023–2025) and shows no sign of stabilising.

Several open research questions follow directly from this review. First, the instruction–data separation problem identified by Liu et al. (2024) remains architecturally unresolved: current mitigations operate around this weakness

through filtering and rails rather than eliminating it, and research into model architectures or training objectives that enforce a genuine boundary between instructions and retrieved data would address the root cause rather than its symptoms. Second, the specific interaction between indirect prompt injection and tool-calling or agentic systems—an area Greshake et al. (2023) flagged as an emerging risk—warrants further empirical study as Azure-based agentic applications built on function calling and Logic Apps become more autonomous and are granted broader permissions. Third, comparative empirical evaluation of Azure-specific controls, such as Prompt Shields, against the benchmarks developed by Liu et al. (2024) would help practitioners quantify the residual risk of a given deployment configuration rather than relying on vendor assurances alone.

7. Conclusion

Prompt injection is a structural consequence of how large language models process instructions and data through a single natural-language channel, and it poses a direct threat to Azure-hosted LLM applications wherever user input, retrieved documents, web content, or tool outputs are combined into a model's context. This paper has organised the technical literature on prompt injection into a six-category taxonomy, mapped this attack surface onto the specific components of an Azure-based architecture—Azure OpenAI Service, Azure AI Search, Azure AI Content Safety, and plugin or Logic Apps integrations—and reviewed the corresponding defense mechanisms and their limitations. The evidence reviewed indicates that no single Azure-native control, whether content filtering, programmable guardrails, or permission scoping, is sufficient in isolation; each addresses a different point in the attack surface and must be combined with the others as part of a defense-in-depth strategy. Given that the underlying vulnerability is unlikely to be fully resolved by model scaling alone, organisations deploying LLM applications on Azure should treat prompt injection defense as an ongoing engineering and monitoring discipline rather than a one-time configuration task.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers

Artificial Intelligence (AI) Use Disclosure: The authors declare that no artificial intelligence tools were used in the preparation of this manuscript.

References

- [1] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 308–318). Association for Computing Machinery. <https://doi.org/10.1145/2976749.2978318>
- [2] Barati, M., Aujla, G. S., Llanos, J. T., Duodu, K. A., Rana, O., Carr, M., & Ranjan, R. (2021). Privacy-aware cloud auditing for GDPR compliance verification in online healthcare. *IEEE Transactions on Industrial Informatics*, 18(7), 4808–4819. <https://doi.org/10.1109/TII.2021.3100152>
- [3] Bernstein, D. (2014). Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1(3), 81–84. <https://doi.org/10.1109/MCC.2014.51>
- [4] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners (arXiv:2005.14165). arXiv. <https://doi.org/10.48550/arXiv.2005.14165>
- [5] Fernandez, E. B., & Monge, R. (2014). A security reference architecture for cloud systems. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems (RACS '14) (pp. 1–5). Association for Computing Machinery. <https://doi.org/10.1145/2578128.2578229>
- [6] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples (arXiv:1412.6572). arXiv. <https://doi.org/10.48550/arXiv.1412.6572>
- [7] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISeC '23) (pp. 79–90). Association for Computing Machinery. <https://doi.org/10.1145/3605764.3623985>

- [8] Hui, B., Yuan, H., Gong, N., Burlina, P., & Cao, Y. (2024). PLeak: Prompt leaking attacks against large language model applications. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24) (pp. 1–16). Association for Computing Machinery. <https://doi.org/10.1145/3658644.3670370>
- [9] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access, 11, 31866–31879. <https://doi.org/10.1109/ACCESS.2023.3262138>
- [10] Li, M. Q., & Fung, B. C. M. (2025). Security concerns for large language models: A survey (arXiv:2505.18889). arXiv. <https://doi.org/10.48550/arXiv.2505.18889>
- [11] Liu, F. W., & Hu, C. (2024). Exploring vulnerabilities and protections in large language models: A survey (arXiv:2406.00240). arXiv. <https://doi.org/10.48550/arXiv.2406.00240>
- [12] Liu, Y., Jia, Y., Geng, R., Jia, J., & Gong, N. Z. (2024). Formalizing and benchmarking prompt injection attacks and defenses (arXiv:2310.12815). arXiv. <https://doi.org/10.48550/arXiv.2310.12815>
- [13] Perez, F., & Ribeiro, I. (2022). Ignore previous prompt: Attack techniques for language models (arXiv:2211.09527). arXiv. <https://doi.org/10.48550/arXiv.2211.09527>
- [14] Rebedea, T., Dinu, R., Sreedhar, M. N., Parisien, C., & Cohen, J. (2023). NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 431–445). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2023.emnlp-demo.40>
- [15] Şaşal, S., & Can, Ö. (2025). Prompt injection attacks on large language models: Multi-model security analysis with categorized attack types. In Proceedings of the 17th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2025), Vol. 1: KDIR (pp. 517–524). SCITEPRESS. <https://doi.org/10.5220/0013838400004000>
- [16] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks (arXiv:1312.6199). arXiv. <https://doi.org/10.48550/arXiv.1312.6199>
- [17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need (arXiv:1706.03762). arXiv. <https://doi.org/10.48550/arXiv.1706.03762>
- [18] Wei, A., Haghtalab, N., & Steinhardt, J. (2023). Jailbroken: How does LLM safety training fail? (arXiv:2307.02483). arXiv. <https://doi.org/10.48550/arXiv.2307.02483>
- [19] Zhang, C., Jin, M., Yu, Q., Liu, C., Xue, H., & Jin, X. (2024). Goal-guided generative prompt injection attack on large language models. In 2024 IEEE International Conference on Data Mining (ICDM) (pp. 941–946). IEEE. <https://doi.org/10.1109/ICDM59182.2024.00119>
- [20] Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J. Z., & Fredrikson, M. (2023). Universal and transferable adversarial attacks on aligned language models (arXiv:2307.15043). arXiv. <https://doi.org/10.48550/arXiv.2307.15043>