Received: 16.08.2026
Accepted: 19.09.2026
Published: 28.10.2026

УДК 004.9+004.942

DOI: 10.32743/UniTech.2026.151.10.23447

Main threats to frontend applications with LLM integration and risk mitigation methods

Yakovleva Margarita
Senior Angular Frontend Developer, V4Scale
Russia, Saint Petersburg
E-mail: margarita.iiakovleva@gmail.com

Основные угрозы для фронтенд-приложений с LLM-интеграцией и способы снижения рисков

Яковлева Маргарита Александровна
ведущий фронтенд-разработчик (Angular), V4Scale
Российская Федерация, г. Санкт-Петербург

Abstract

The article examines security threats to frontend applications integrated with large language models. The aim of the study is to develop a methodological approach to mitigating the risks of LLM integration in the frontend, taking into account both classical web threats and new LLM-specific threats. The study demonstrates that the integration of an LLM into the user interface transforms the application's risk profile. The specific features of LLM-integrated frontend applications are revealed, a threat classification is developed, and the differences between classical frontend/web vulnerabilities and threats related to prompt injection, prompt leaking, data exfiltration, RAG poisoning, and unsafe tool calls are identified. Particular attention is paid to prompt injection as a threat to LLM integration that can lead to the bypassing of system rules, leakage of user and system data, distortion of model responses, and execution of undesirable actions through connected tools. A typical risk mitigation algorithm is proposed, including procedures for inventorying LLM functions, classifying data, separating instructions from user input, limiting model permissions, validating responses, logging, and red-team testing. The practical significance of the study lies in the applicability of the proposed approach to the design and maintenance of frontend applications with LLM interfaces.

Библиографическое описание: Яковлева М.А. Основные угрозы для фронтенд-приложений с LLM-интеграцией и способы снижения рисков // Universum: технические науки : электрон. научн. журн. 2026. 10(151). URL: https://7universum.com/ru/tech/archive/item/23447

For citation: Yakovleva M. Main threats to frontend applications with LLM integration and risk mitigation methods // Universum: Technical Sciences : electronic scientific journal. 2026. No. 10(151). Available at: https://7universum.com/ru/tech/archive/item/23447

Аннотация

В статье рассматриваются угрозы безопасности для фронтенд-приложений с интеграцией больших языковых моделей. Цель исследования состоит в разработке методического подхода к снижению рисков LLM-интеграции во фронтенде с учетом классических веб-угроз и новых угроз, специфичных для LLM. Обосновано, что включение LLM в пользовательский интерфейс трансформирует профиль риска приложения. Раскрыта специфика фронтенд-приложений с LLM-интеграцией, сформирована классификация угроз, выделены различия между классическими уязвимостями фронтенд- и веб-приложений и угрозами, связанными с prompt injection, prompt leaking, data exfiltration, RAG poisoning и unsafe tool calls. Особое внимание уделено prompt injection как угрозе LLM-интеграции, способной приводить к обходу системных правил, утечке пользовательских и служебных данных, искажению ответа модели и выполнению нежелательных действий через подключенные инструменты. Предложен типовой алгоритм снижения рисков, в который входят процедуры инвентаризации LLM-функций, классификации данных, разделения инструкций и пользовательского ввода, ограничения операций модели и подключенных инструментов, проверки ответов модели, логирования и red-team тестирования. Практическая значимость исследования состоит в возможности применения предложенного подхода при проектировании и сопровождении фронтенд-приложений с LLM-интерфейсами.

Keywords: frontend application, LLM, prompt injection, prompt leaking, web security, frontend security, data leakage, RAG poisoning, unsafe tool calls, risk-based approach.

Ключевые слова: фронтенд-приложение, LLM, prompt injection, prompt leaking, web security, frontend security, data leakage, RAG poisoning, unsafe tool calls, risk-based approach.

Introduction

Modern frontend applications are increasingly equipped with LLM-based interfaces, including chatbots, search assistants, content generation tools, document analysis systems, and data assistants. Such applications process not only forms, tokens, cookies, API requests, and browser data, but also prompts, conversation history, system instructions, model outputs, external documents, and the results of tool calls. As a result, the boundary between user data and control instructions becomes less distinct. Indirect prompt injection exploits this convergence of data and instructions, as a malicious instruction may enter the application through a document, web page, or another external source whose content is subsequently passed to the model [1, pp. 81–82].

The relevance of the study stems from the fact that the frontend layer of an application often serves as the user's first point of interaction with an LLM, as it is responsible for collecting user input, displaying model responses, initiating API requests, and transmitting session-related information. A vulnerability at this layer may result in the leakage of user data, disclosure of system instructions, unauthorized tool invocation, or the generation of unsafe responses. Therefore, these risks should be taken into account during the application design process.

The aim of the study is to develop a methodological approach to mitigating the risks associated with LLM integration in frontend applications.

Research methodology

The study is based on a comparative analysis of the scientific literature on the security of LLM-integrated applications, as well as studies on prompt injection, prompt leaking, Prompt-to-SQL injection, structured queries, and the Instruction Hierarchy. In addition, a scenario-based analysis of a typical frontend application with an LLM interface was conducted.

The object of analysis is a web application in which the user interacts with an LLM through a web browser. The analysis covers user request submission, data collection from the user interface (UI), prompt construction, transmission of the request to the backend, communication between the backend service and the LLM, presentation of the model response, as well as the potential execution of actions through APIs, retrieval-augmented generation (RAG), or the tool layer.

The overall risk associated with an LLM-integrated frontend application can be expressed as follows:

$$R_{total} = R_{web} + R_{LLM} + R_{data} + R_{tool},$$

where R_{web} represents classical web security threats; R_{LLM} represents threats associated with prompt injection, jailbreak, prompt leaking, and output manipulation; R_{data} represents the risk of user and system data leakage; R_{tool} represents the risk of undesirable actions performed through APIs, RAG, plugins, and external services.

For practical risk assessment, a five-point scale is used for two parameters: likelihood and impact. The overall risk priority is determined as the product of these parameters:

$$Risk\ Score = L * I,$$

where L denotes the likelihood of the threat occurring, and I represents the severity of its impact.

Results and discussion

An LLM-integrated frontend application differs from a conventional web interface in that part of its behavior is determined by the processing of natural language. The user enters a free-form request, which the model interprets in relation to the system instructions and uses to generate a response. When tools are connected, the model output may be used by the application to call an API, search a knowledge base, generate an SQL query, send a message, or modify data.

The distinctive characteristics of an LLM-integrated frontend application can be described by four key features: open-ended user input, the presence of system instructions, the processing of external data, and the ability to invoke tools (tool calls). Users may provide the model not only with data but also with malicious instructions intended to circumvent established safeguards. The system prompt, developer instructions, and processing rules may contain information about the application's internal logic. In addition, documents, web pages, emails, and RAG data sources may include hidden instructions, while connected tools can transform an incorrect interpretation of such instructions into an operational risk.

Based on the above analysis, the following classification of threats to LLM-integrated frontend applications has been developed:

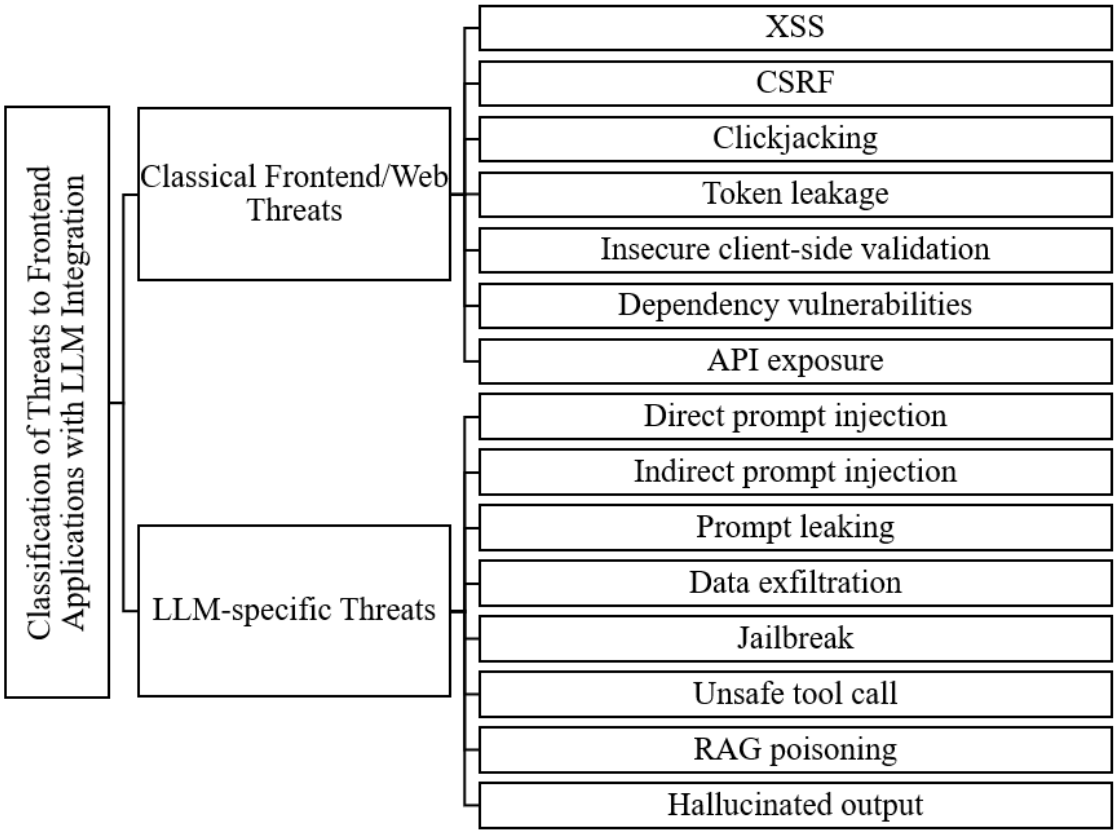


Figure 1. Classification of threats to LLM-integrated frontend applications. Source: compiled by the author

A comparison of the two broad categories of threats reveals the following key differences (Table 1):

Table 1. Comparison of conventional threats to frontend and web applications with threats specific to LLM-integrated applications

Criterion	**Conventional frontend and web application threats**	**LLM-specific threats**
Primary source	Application code, DOM, APIs, dependencies, browser storage	Prompts, external text, RAG sources, model responses
Attack mechanism	Exploitation of technical vulnerabilities	Manipulation of model behavior through textual instructions
Typical example	XSS, CSRF, token leakage	Prompt injection, prompt leaking
Protected assets	Frontend code, user sessions, tokens, API requests	System prompts, user data, application policies, tool calls
Primary security controls	SAST, DAST, Content Security Policy (CSP), dependency scanning	Prompt testing, output validation, red-team testing
Typical impact	Session hijacking, arbitrary code execution, token leakage	Policy bypass, prompt disclosure, unauthorized actions
Security control layer	Frontend, backend, HTTP headers, browser storage	Prompt layer, policy layer, tool layer, logging

One of the key security threats associated with LLM integration is prompt injection. In a direct prompt injection attack, the user explicitly submits an instruction intended to alter the model's behavior. In an indirect prompt injection attack, the malicious instruction is embedded in external content, such as a document, web page, email, knowledge base, or text retrieved from a third-party

source. According to the study in [2, p. 1], prompt injection can lead to prompt theft, unauthorized use of LLM capabilities, and the circumvention of application safeguards. The proposed HouYi attack method was evaluated on 36 real-world LLM-integrated applications, of which 31 were found to be vulnerable [2]. Figure 2 illustrates the attack chain through an LLM interface.

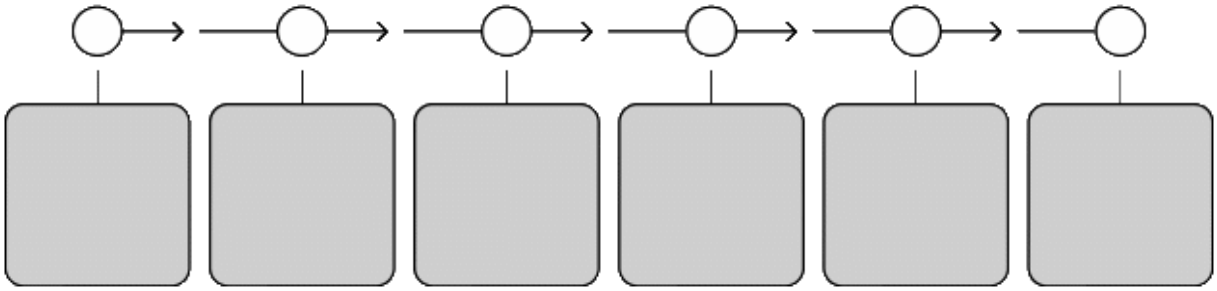


Figure 2. Prompt injection attack chain through an LLM interface. Source: compiled by the author

Thus, the attack chain begins with untrusted input, such as a user message, external document, web page content, a fragment retrieved from a RAG source, or the output of another tool. A prompt is then constructed at the frontend or backend layer, where user-provided data may be combined with system instructions. If the model does not reliably distinguish between trusted instructions and the data being processed, it may incorrectly interpret a malicious command embedded in the data as an instruction to execute. When the attack succeeds, the command affects either the generated response or the selection of a subsequent action. This may result in the disclosure of system instructions, exposure of another user's data, generation of malicious code, manipulation of recommendations, or an unauthorized call to an external tool.

LLM resilience to prompt injection is commonly evaluated using benchmark-based approaches. Such attacks can be tested against large collections of adversarial and defensive prompts, making a single manual assessment insufficient to establish the security of an LLM interface [3, pp. 1–3]. Another important threat is prompt leaking, an attack aimed at extracting the system prompt, which may contain processing rules, internal constraints, response templates, and information about the application's security policies [4, p. 3600]. Web applications are also vulnerable to Prompt-to-SQL injection, in which a user prompt influences the generation of SQL queries. Accordingly, effective protection should include the principle of least privilege, validation of generated SQL queries, and verification of database operations before execution [5, p. 1778]. Based on the above analysis, a general algorithm for mitigating the risks associated with LLM integration can be proposed (Figure 3)

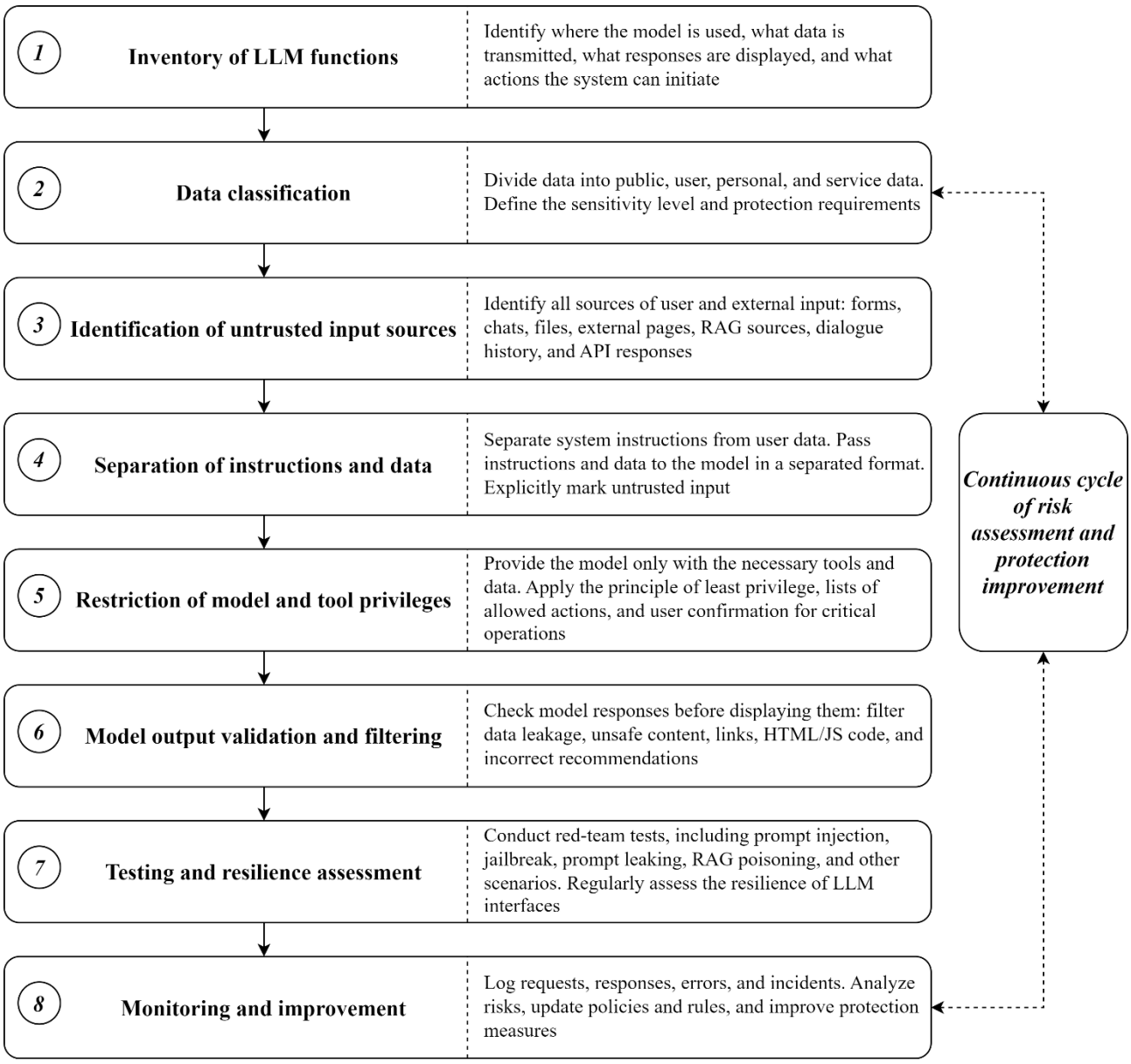


Figure 3. General algorithm for mitigating the risks of LLM integration in frontend applications. Source: compiled by the author

The standard algorithm for mitigating the risks of LLM integration is structured as a sequential procedure for controlling data, prompts, the model, tools, and the user interface. It begins with an inventory of LLM-enabled functions, data classification, and the identification of untrusted input sources. The next step involves separating system instructions from user-provided data in accordance with the StruQ approach [6, p. 2383]. Subsequent stages include restricting the model's permissions, validating outputs, controlling tool calls, conducting red-team testing, and monitoring security incidents. Critical actions initiated through an LLM must undergo server-side validation, be restricted by an allowlist and the principle of least privilege, and require user confirmation. Such actions should be executed by the application or the tool layer rather than by the model itself. Before being displayed, model responses must undergo output validation and checks for personal data leakage, while HTML and Markdown content must be sanitized. In addition, instruction priority should be explicitly defined so that system instructions and developer instructions take

precedence over user input and third-party data. This principle is consistent with the Instruction Hierarchy approach and reduces the risk of internal rules being overridden by untrusted content [7, p. 2].

Practical implementation of the proposed algorithm requires that secrets never be included in prompts, user input be explicitly identified as untrusted, and API and tool calls be executed through a backend service with appropriate authorization checks [8, p. 10471]. Security measures specific to the LLM layer complement, rather than replace, conventional frontend and web application security practices. Accordingly, Content Security Policy, secure token storage, dependency management, protection against XSS attacks, and validation of generated HTML and Markdown content remain essential. The decision to enable an LLM-powered function should be based on an assessment of the residual risk, particularly when the function involves personal data, financial transactions, internal documents, or legally significant operations.

To summarize the proposed approach, Figure 4 presents a conceptual assessment of risk levels before and after the implementation of protective measures. It highlights differences between threat categories, the relationship between likelihood and impact, the effectiveness of security controls, and the distribution of risks across the different layers of the application

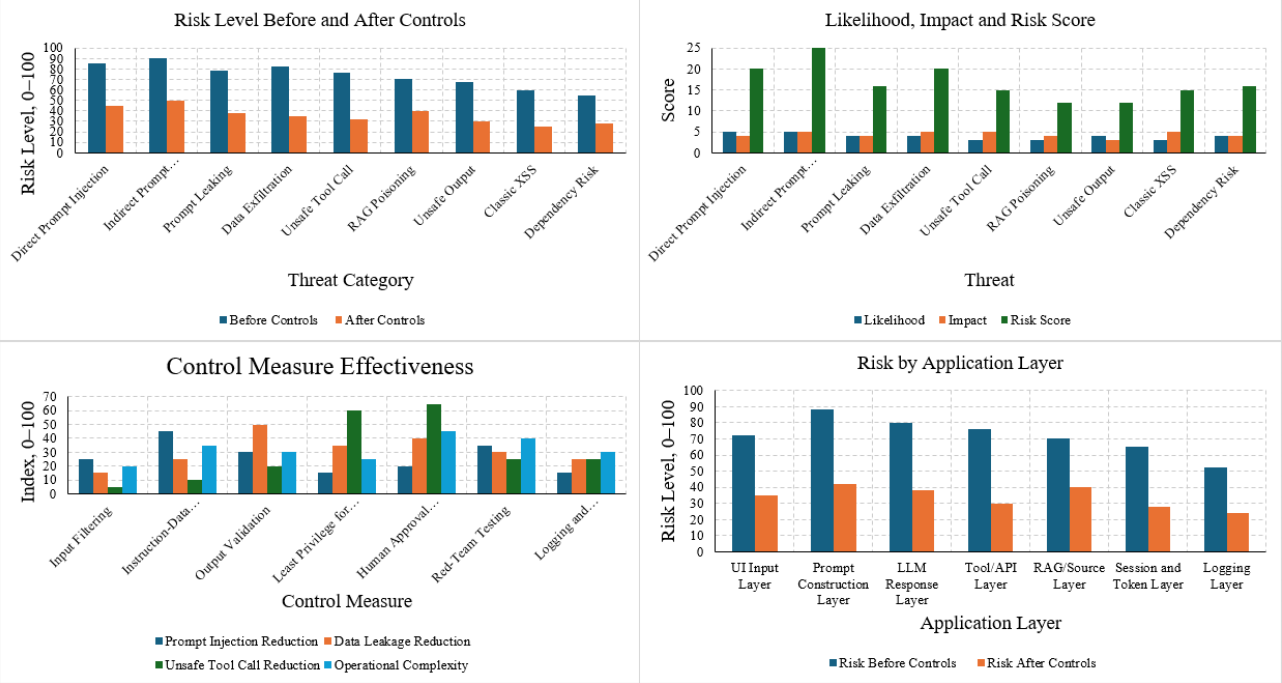


Figure 4. Comparison of risk levels for an LLM-integrated frontend application before and after the implementation of security measures. Source: compiled by the author

The results indicate that the highest baseline risk is associated with indirect prompt injection, direct prompt injection, data exfiltration, and prompt leaking. Within the proposed model, the level of risk decreases across all threat categories following the implementation of security measures; however, residual risk remains for indirect prompt injection and RAG poisoning [9, p. 3827]. Jailbreak attacks constitute another relevant threat, as specially constructed user instructions may be used to circumvent the behavioral restrictions imposed on the model [10, p. 1486]. Particular attention should be paid to the prompt construction layer, as it is the point at which user input, system instructions, and application data are combined.

Conclusion

Frontend applications with LLM integration require a dedicated risk-oriented approach to security. While conventional web security threats remain relevant, they are supplemented by risks specific to LLM-based systems, including prompt injection, prompt leaking, data exfiltration, unsafe tool calls, and RAG poisoning. Prompt injection should be regarded as one of the most critical threats because it exploits the model's ability to follow natural language instructions and may enable the circumvention of application security policies. The scientific contribution of this study lies in distinguishing between these two categories of threats and in developing an algorithm for mitigating the risks associated with LLM integration. The practical value of the proposed approach lies in its application to the design, development, and maintenance of LLM-based interfaces.

References

1. Greshake K., Abdelnabi S., Mishra S., Endres C., Holz T., Fritz M. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection // Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security. ACM. – 2023. – P. 79–90. – DOI: 10.1145/3605764.3623985.
2. Liu Y., Deng G., Li Y., Wang K., Zhang T., Liu Y., Wang H., Zheng Y., Liu Y. Prompt Injection attack against LLM-integrated Applications // arXiv preprint arXiv:2306.05499v1. – 2023. – P. 1–18. – DOI: 10.48550/arXiv.2306.05499. – URL: https://arxiv.org/abs/2306.05499v1 (accessed: 19.08.2026).
3. Toyer S., Watkins O., Mendes E.A., Svegliato J., Bailey L., Wang T., Ong I., Elmaaroufi K., Abbeel P., Darrell T., Ritter A., Russell S. Tensor Trust: Interpretable Prompt Injection Attacks from an Online Game // Proceedings of the International Conference on Learning Representations. – 2024. – P. 1–33.
4. Hui B., Yuan H., Gong N., Burlina P., Cao Y. PLeak: Prompt Leaking Attacks against Large Language Model Applications // CCS '24: Proceedings of the. – New York: Association for Computing Machinery, 2024. – P. 3600–3614. – DOI: 10.1145/3658644.3670370. – URL: https://dl.acm.org/doi/10.1145/3658644.3670370 (accessed: 19.08.2026).
5. Pedro R., Castro D., Carreira P., Santos N. Prompt-to-SQL Injections in LLM-Integrated Web Applications: Risks and Defenses // Proceedings of the 47th IEEE/ACM International Conference on Software Engineering. IEEE. – 2025. – P. 1768–1780. – DOI: 10.1109/ICSE55347.2025.00007.
6. Chen S., Piet J., Sitawarin C., Wagner D. StruQ: Defending Against Prompt Injection with Structured Queries // Proceedings of the 34th USENIX Security Symposium. USENIX Association. – 2025. – P. 2383–2400.
7. Wallace E., Xiao K., Leike R., Weng L., Heidecke J., Beutel A. The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions // arXiv preprint arXiv:2404.13208. – 2024. – P. 1–13. – DOI: 10.48550/arXiv.2404.13208. – URL: https://arxiv.org/abs/2404.13208 (accessed: 19.08.2026).

8. Zhan Q., Liang Z., Ying Z., Kang D. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents // Findings of the Association for Computational Linguistics: ACL. – 2024. – P. 10471–10506. – DOI: 10.18653/v1/2024.findings-acl.624. – URL: https://aclanthology.org/2024.findings-acl.624/ (accessed: 12.09.2026).
9. Zou W., Geng R., Wang B., Jia J. PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models // Proceedings of the 34th USENIX Security Symposium. – 2025. – P. 3827–3844. – URL: https://www.usenix.org/conference/usenixsecurity25/presentation/zou-poisonedrag (accessed: 12.09.2026).
10. Xie Y., Yi J., Shao J., Curl J., Lyu L., Chen Q., Xie X., Wu F. Defending ChatGPT against Jailbreak Attack via Self-Reminders // Nature Machine Intelligence. – 2023. – Vol. 5, No. 12. – P. 1486–1496. – DOI: 10.1038/s42256-023-00765-8. – URL: https://www.nature.com/articles/s42256-023-00765-8 (accessed: 12.09.2026).