середньоквадратичного відхилення дало змогу кількісно оцінити стабільність інференсу та виміряти рівень джиттеру в динамічному відео потоці.

Результати роботи методу HOG (CPU). Апробація класичного детектора на базі CPU продемонструвала високу прогнозованість часових показників. Середня латентність обробки кадру склала 25-30 мс (30-40 FPS), що варіюється залежно від вхідної роздільної здатності. Головною перевагою методу HOG визначено економічну доступність та відсутність потреби у спеціалізованих обчислювачах. Водночас виявлено суттєве обмеження точності: алгоритм втрачає працездатність при значних ракурсних відхиленнях обличчя та в умовах динамічної зміни освітленості.

Результати роботи методу CNN (GPU/CUDA). Використання CNN-детектора без апаратного прискорення виявило критичну нестачу продуктивності: затримка інференсу понад 800 мс (~1.2 FPS) робить застосування центрального процесора (CPU) недоцільним для систем реального часу. Впровадження технології NVIDIA CUDA забезпечило радикальний приріст швидкодії, скоротивши час обробки до 15–20 мс, що відповідає частоті понад 50 FPS. Окрім високої продуктивності, нейромережевий підхід продемонстрував вищу повноту детекції (recall), стабільно ідентифікуючи обличчя в складних ракурсах, де класичний метод HOG виявився неефективним

Порівняльна характеристика. Отримані дані підтверджують ефективність залучення GPU-ресурсів: показник інтенсифікації обчислень (speedup) склав 2,28 відносно базової архітектури.

Висновок. У роботі проведено оцінку та оптимізацію інструментів dlib для детекції облич. Доведено, що обмежена продуктивність CPU не дозволяє використовувати нейромережі (CNN) у режимі реального часу. Натомість впровадження технології NVIDIA CUDA забезпечило кратне прискорення: GPU-орієнтований CNN-підхід перевершив класичний метод HOG як за точністю, так і за швидкістю інференсу.

Security vulnerabilities at the Python LLM frameworks boundary

UDC 004.896:004.056.5

Oleksandr Karnaukhov[1], Nataliya Zagorodna[2], Oleh Yarema[3], Oleksandr Revniuk[4]

Ternopil Ivan Puluj National Technical University,
[1]karnaukhov@live.com, [2]zagorodna_n@tntu.edu.ua, [3]yarema.oleh.m@gmail.com, [4]revo0708@gmail.com

To execute complex workflows such as dynamic data analysis, file system management, and database querying, popular Python orchestration frameworks, including LangChain, LlamaIndex, and Ollama, frequently grant LLMs direct access to runtime environments [1]. This architecture often relies on underlying Python utilities like “exec()” or “eval()” to translate model outputs into system actions.

However, this design introduces a critical conflict between traditional deterministic software security and the probabilistic nature of linguistic models. While standard application security relies on strict input sanitization at the system boundary, AI agents inherently process untrusted natural language payloads directly from users.

This problem gives rise to “prompt injection” vulnerabilities, where an attacker makes malicious linguistic instructions designed to override the agent’s core system prompts [2].

The fundamental security flaw is at the structural intersection where natural language instructions transition into machine-executable Python code. If the orchestration framework lacks strict isolation or sanitization layers, a successful prompt injection can easily escape the application context [3]. This boundary effectively transforms a simple text interface into a vector for exploitation, with potential to escalate linguistic manipulation into unauthorized system command execution, file system traversal, or arbitrary Remote Code Execution (RCE) via Python’s subprocess or OS modules. Consequently, parsing errors and context window exploitation within Python ecosystems present a risk to production-grade AI deployments [4].

Table 1

Security architecture comparison

Dimension	Traditional Python Applications	Emerging Python LLM Agents
Input type	Deterministic - structured data, strings, integers	Probabilistic - unstructured natural language
Security boundary	Strict input sanitization	Open text interface directly exposed to user
Execution mechanism	Pre-defined, compiled, or hardcoded logic functions	Dynamic translation of text into code using “exec()” or “eval()”
Primary threat vector	SQL injection, buffer overflows, malicious code inputs	Prompt injection, linguistic manipulation overriding rules
Highest risk impact	Application crashes, data leaks via specific bugs	Remote Code Execution (RCE) via subprocess
Proposed mitigation	Standard input validation and parameterized queries	Abstract Syntax Tree (AST) validation prior to execution

To address these risks, an empirical, sandbox-based experimental framework to quantify the vulnerability threshold of current Python LLM orchestration layers is proposed. By constructing a prototype AI agent, we will subject both a baseline Python implementation and a standardized framework deployment to a curated benchmark suite of distinct prompt injections. These injections will explicitly target the boundary where natural language triggers backend execution, attempting to make the agent perform unauthorized operations. The experiment can evaluate framework resilience by measuring injection success rates and mapping the specific architectural blind spots where linguistic contexts successfully override Python-level security constraints. Experimental findings are expected to reveal deficiencies in native input sanitization, providing concrete data to advocate for deterministic mitigation strategies prior to code execution.

1. Chen, Y.-J., & Madisetti, V. K. (2025). Information security, ethics, and integrity in LLM agent interaction. Journal of Information Security, 16(01), 184–196. https://doi.org/10.4236/jis.2025.161010
2. Lee, D., Tiwari, M., & Miranda, B. (2026). Prompt infection: Llm-to-llm

prompt injection within multi-agent systems. У Lecture notes in computer science (с. 511–520). Springer Nature Switzerland. https://doi.org/10.1007/978-3-032-16092-8_28

3. AlSobeh, A., Gwarzo, Z., & Shatnawi, A. (2025). ShadowPlay: Engineering defenses against role-based prompt injection and dependency hallucination in llm-powered development. У 2025 international conference on cybersecurity and ai-based systems (cyber-ai) (с. 317–325). IEEE. https://doi.org/10.1109/cyber-ai66431.2025.11233258
4. Shi, J., Yuan, Z., Tie, G., Zhou, P., Gong, N., & Sun, L. (2026). Prompt injection attack to tool selection in LLM agents. У Network and distributed system security symposium. Internet Society. https://doi.org/10.14722/ndss.2026.230675

Метод попарного порівняння AHP для пріоритезації безпекових контролів SSDF у CI/CD

УДК 004.056:004.4

Тарас Лечаченко[1], Дмитро Войтович[2]

Тернопільський національний технічний університет імені Івана Пулюя,
[1]lechachenko.taras@tntu.edu.ua, [2]voitovuch855@gmail.com

У зв'язку зі зростанням кількості кіберзагроз і складністю CI/CD-інфраструктур особливої актуальності набуває завдання пріоритизації безпекових контролів для DevSecOps та CI/CD-середовищ із подальшим кількісним оцінюванням їхньої ефективності. Існуючі підходи переважно зосереджуються на виявленні загроз або описі окремих механізмів захисту, однак недостатньо уваги приділяється формалізованим методам оцінювання та ранжування безпекових заходів відповідно до рівня критичності загроз.Як зазначають автори роботи [1], за останні п'ять років кількість досліджень у сфері DevSecOps суттєво зросла, проте питання пріоритезації безпекових контролів у контексті конкретних загроз для CI/CD-інфраструктури залишається недостатньо дослідженим.

Одним із перспективних підходів до розв'язання цієї задачі є застосування методу аналізу ієрархій (AHP, Analytic Hierarchy Process) [2], який базується на попарному порівнянні критеріїв та альтернатив. Використання AHP дозволяє формалізувати процес прийняття рішень, визначити вагомість окремих загроз і безпекових контролів, а також забезпечити обґрунтовану пріоритизацію заходів захисту для DevSecOps та CI/CD-середовищ. Для ранжування заходів захисту проти загроз CI/CD за основу взято STRIDE-методологію та визначено, що серед шести категорій STRIDE три в контексті CI/CD є першочерговими: підробка коду та артефактів, підміна сутності та розкриття інформації. Безпекові контролі для ідентифікації засобів захисту конвеєрів CI/CD було взято з NIST Secure Software Development Framework (SSDF) методології [3], оскільки вона забезпечує орієнтовані на життєвий цикл практики, що відповідають DevSecOps. Для забезпечення точності та практичної доцільності пріоритезації з переліку контролів SSDF було обрано підмножину з десяти засобів контролю: