Institut Riset dan Publikasi Indonesia (IRPI)
MALCOM: Indonesian Journal of Machine Learning and Computer Science
Journal Homepage: https://journal.irpi.or.id/index.php/malcom
Vol. 6 Iss. 2 April 2026, pp: 973-985
ISSN(P): 2797-2313 | ISSN(E): 2775-8575

Intelligent Automation of Linux System Administration via the NeuroSysAI Autonomous Agent: Security and Cost-Efficiency Analysis

Auliya Ur Rahman Ash-Shidqi[1*], Helmy Faisal Muttaqin[2]

[1,2]Department of Computer Science, Widyatama University, Bandung, Indonesia

E-Mail: [1]auliya.urrahman@widyatama.ac.id, [2]helmy.faisal@widyatama.ac.id

Received Feb 16th 2026; Revised Mar 26th 2026; Accepted Apr 20th 2026; Available Online Apr 30th 2026
Corresponding Author: Auliya Ur Rahman Ash-Shidqi
Copyright © 2026 by Authors, Published by Institut Riset dan Publikasi Indonesia (IRPI)

Abstract

Manual Linux system administration frequently suffers from operational inefficiencies and a high risk of human error. As IT infrastructure grows increasingly complex, traditional automation tools often lack the adaptability needed to handle dynamic troubleshooting. To address these limitations, this study introduces NeuroSysAI, an autonomous agent powered by Large Language Models (LLMs) specifically designed to automate server configuration, security management, and system monitoring in Ubuntu 22.04 LTS environments. Our approach implements a hybrid architecture that leverages the Mistral-Nemo API for complex reasoning alongside a local GPT-OSS model via Ollama to optimize operational costs. A primary contribution of this research is the integration of strict tool-use restrictions. This mechanism effectively mitigates the inherent risk of LLM hallucinations, ensuring that terminal command executions remain secure and controlled. Functional validation demonstrates that NeuroSysAI is highly reliable, achieving a 95.8% success rate in resolving administrative tasks with a robust system load capacity of 34 requests per second (RPS). Beyond technical performance, our threat modeling and cost evaluations confirm that this hybrid agent approach provides an optimal balance between infrastructure investment (CapEx) and operational efficiency (OpEx). Ultimately, NeuroSysAI offers system administrators a secure, adaptive, and economically viable solution for modernizing IT operations.

Keywords: Linux System, Artificial Intelligence, Automation, Natural Language Processing, Thread Modeling

1. INTRODUCTION

Linux servers form the critical backbone of modern IT infrastructures, handling everything from web hosting to complex enterprise databases. However, managing these environments requires highly specialized expertise. Traditionally, administrators have relied on manual command-line execution or imperative Infrastructure-as-Code (IaC) tools like Ansible to mitigate human error [1], [2]. The shift toward Artificial Intelligence for IT Operations (AIOps) has recently become essential for managing the increasing complexity of distributed failure management and monitoring [3].

In recent years, the industry has shifted toward LLM-integrated AIOps to optimize and secure modern software delivery [4], [5]. At the forefront of this shift is the application of Large Language Models (LLMs). Models like GPT-4 have demonstrated "sparks" of general intelligence, showing human-level performance in coding and technical troubleshooting without the need for specialized prompting [6]. This capability is foundational for the development of autonomous agents that can act as human-like decision-makers in engineering environments [7].

Despite these promising advancements, existing implementations expose a critical research gap regarding operational security and cost-efficiency. As highlighted by [8], LLMs inherently suffer from "hallucinations" generating plausible but entirely incorrect outputs. When applied to system administration, an AI hallucination can lead to the execution of destructive terminal commands. Furthermore, studies assessing AI-generated code contributions warn about significant vulnerabilities and unauthorized privilege escalations when AI operates without strict boundaries [9]. Additionally, relying exclusively on premium, cloud-based models like GPT-4 (as seen in Cao et al.'s approach) introduces high operational expenses (OpEx) and potential data privacy concerns, making it less viable for scalable, long-term enterprise deployment.

Therefore, a critical need exists for an intelligent automation tool that bridges the gap between dynamic LLM capabilities and strict server security, without incurring prohibitive cloud API costs.

To address this gap, this study introduces NeuroSysAI, an autonomous agent specifically designed for secure Linux system administration within Ubuntu 22.04 LTS environments. This research differentiates itself from previous studies in two fundamental ways:

1. Hybrid Architecture for Cost-Efficiency: Unlike previous approaches that depend entirely on expensive cloud models, NeuroSysAI implements a hybrid reasoning engine. It utilizes the Mistral-Nemo API for complex, ad-hoc troubleshooting, while offloading routine execution tasks to a local GPT-OSS model via Ollama. This approach significantly reduces token costs and enhances data privacy.
2. Strict Tool-Use Restrictions for Security: Instead of granting the AI direct, unrestricted shell access, NeuroSysAI operates through a rigidly defined "tool-call" mechanism. The AI is confined to generating specific JSON-formatted execution intents, which the backend validates before triggering any actual Bash commands, thereby mitigating the catastrophic risks of LLM hallucinations.

By evaluating functional success rates, system load capacity, and comprehensive threat models, this paper demonstrates that NeuroSysAI provides a secure, adaptive, and economically viable alternative to both traditional manual administration and fully cloud-dependent AI agents.

2. MATERIALS AND METHOD

To ensure a structured and scientifically rigorous approach in developing NeuroSysAI, this research is divided into two primary sections: establishing the theoretical foundations derived from a comprehensive literature review and executing a systematic research workflow.

2.1. Theoretical Studies and Literature Review

The evolution of IT infrastructure demands highly efficient administration to maintain business continuity [10], [11]. Applying machine learning from an infrastructure perspective is critical for maintaining performance in large-scale datacenters [12]. The practical implementation of these systems is heavily supported by modern libraries like Transformers, which provide state-of-the-art natural language processing capabilities for automated tasks [13].

To minimize logical errors and handle exceptions during execution, NeuroSysAI adopts a synergy between reasoning and acting, known as the ReAct paradigm [14]. This allows the agent to induce, track, and update action plans while interfacing with the Linux terminal environment. However, granting AI direct access to servers introduces risks. Mitigating these risks requires aligning models with user intent through human feedback, ensuring that AI-driven agents follow instructions safely and accurately [15].

Today, this paradigm is shifting toward AI-Augmented DevOps and AIOps (Artificial Intelligence for IT Operations) to optimize and secure modern software delivery [5], [16], [17]. In complex architectures, such as microservices, AIOps has proven highly capable of performing context-aware anomaly detection[18]. At the core of this intelligent automation is the Transformer architecture[19], which has revolutionized both natural language processing and machine learning-based code generation [20]. Recent systematic reviews highlight the expansive potential of Large Language Models (LLMs) in software engineering [21]. Specifically, models utilizing structure-aware transformers, such as StructCoder, demonstrate high precision in translating human intent into executable scripts [22]. Building on this, recent empirical evaluations have proven that LLM-based AI agents, including GPT-4 variants, can effectively manage Linux servers [23].

However, granting autonomous AI direct execution access to production servers introduces critical cybersecurity vulnerabilities. Recent surveys emphasize the severe risks posed by LLM hallucinations [8] and the inherent security flaws often found in AI-generated code contributions, such as those from GitHub Copilot [9] In the context of national and regional cyber threats, as highlighted by the ID-SIRTII/CC cybersecurity reports [24] mitigating these risks requires robust cloud security tools and remediation strategies [25]. Therefore, the engineering of NeuroSysAI adheres strictly to established software engineering principles [26], [27] The development lifecycle integrates a comprehensive security testing framework suitable for the Secure Software Development Life Cycle [28], applies systematic security metrics for cyber-physical systems[29], and incorporates standardized usability evaluation methods [30] to ensure the agent remains both safe and user-centric.

2.2. Research Flow and Methodology

The overall methodology of this study follows a systematic progression designed to ensure the reliability and security of the proposed application, as illustrated in Figure 1. The detailed stages are as follows:

1. Problem Identification and Requirement Analysis: The research was initiated by analyzing the operational bottlenecks and security challenges experienced by system administrators. By evaluating the high risk of human error during manual configurations and reviewing the literature on AIOps

limitations, we formulated the specific technical requirements for a secure, AI-driven automation tool tailored for Ubuntu 22.04 LTS environments.

2. Architectural Design and Tool Selection: Following the requirement analysis, we designed a hybrid system architecture. This involved selecting a Vue.js frontend for intuitive user interaction, a Django backend for core request processing, and a PostgreSQL database to maintain reliable activity logs for auditing purposes. Crucially, we mapped out the decision-making logic required to bridge the AI reasoning engine with the underlying Linux terminal shell safely.
3. AI Integration and System Implementation: During the core development phase, the reasoning engine was programmed using the Mistral-Nemo model (via API) for complex troubleshooting tasks, alongside a local GPT-OSS model (via Ollama) for cost-effective, routine task execution. To address the hallucination risks identified in the literature, we strictly enforced "tool-use restrictions" at the system prompt level, preventing the agent from executing unauthorized or destructive bash commands.
4. Validation and Security Testing: To guarantee system integrity, we subjected the application to rigorous, multi-layered evaluation. Initial functional validation included unit and integration testing to ensure seamless communication between the backend and the AI models. Subsequently, comprehensive security assessments were executed using external penetration tools including NMAP, OWASP ZAP, SSL Labs, and Nuclei to identify vulnerabilities such as prompt injection. Finally, usability testing was conducted with end users to assess interface acceptance.
5. Result Evaluation and Conclusion: In the final stage, all testing data was aggregated and analyzed. We evaluated the functional success rate, measured the system load capacity (Requests Per Second), and synthesized the findings into an OpEx/CapEx cost analysis. This comprehensive evaluation allowed us to draw final conclusions and propose actionable recommendations for future autonomous system administration deployments.

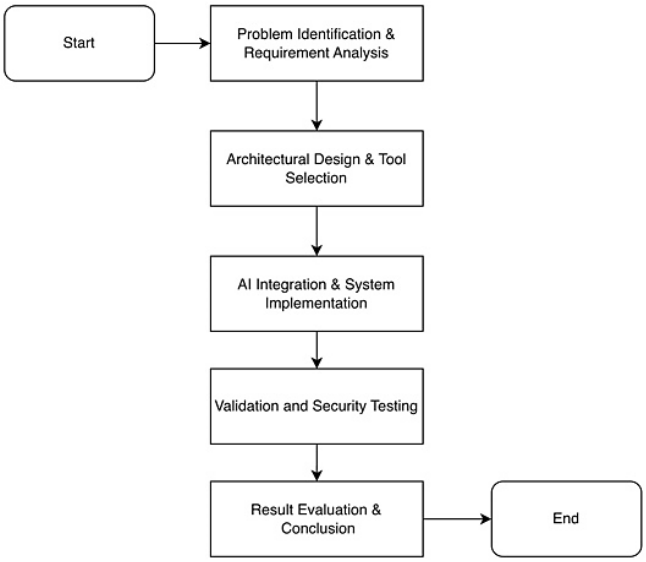


Figure 1. Research Methodology Flowchart

3. RESULTS AND DISCUSSION

Analysis of questionnaires administered to system administrators at PT. X indicates that the majority of administrators feel the need for automation in Linux system administration. AI is considered a potential solution for improving efficiency and reducing human error. The most desired features are configuration automation, error detection, automatic logging, and recommendations based on usage patterns. Despite high enthusiasm, security, stability, and transparency are top priorities. Interest in trying AI applications such as NeuroSysAI is quite high, provided that the system is secure and stable.

The NeuroSysAI application developed in this research is designed as a centralized platform to simplify and automate various aspects of Linux system and network administration. With an architecture that integrates a Django-based backend, Vue.js frontend, and a series of artificial intelligence models, NeuroSysAI provides an intuitive and feature-rich web interface. The overall system architecture, illustrating the data flow from the user interface to the AI decision-making logic and execution, is presented in Figure 2. The application's key features are specifically designed to address the challenges faced by system administrators, from daily monitoring to security incident response.

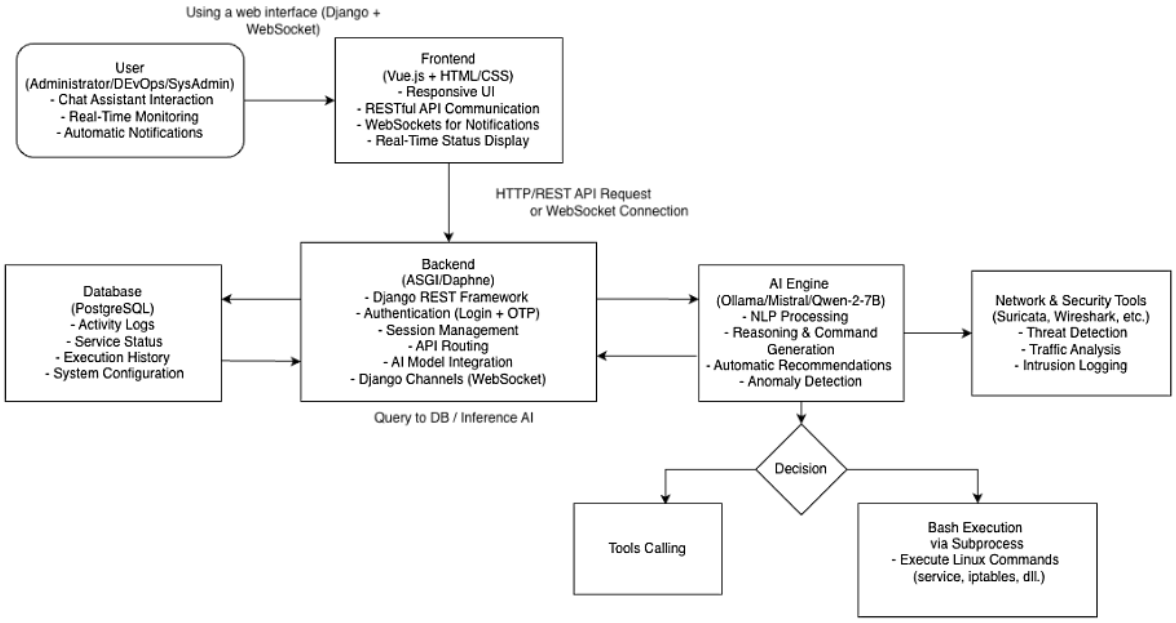


Figure 2. NeuroSysAI System Architecture

During the implementation phase, the NeuroSysAI application was successfully built and tested in a simulated environment that resembled the network infrastructure and Linux operating system used by PT. X. The testing process was conducted thoroughly using quantitative and qualitative approaches, with the aim of ensuring that the system could operate in accordance with technical specifications and meet user functional and security requirements. The system implementation involved various technological components, including a Vue.js-based frontend, Django backend, PostgreSQL database, local and cloud AI models, and integration with network tools such as Suricata and Wireshark. To validate the entire system, testing was carried out in six main stages: unit testing, integration testing, usability testing, security testing, performance testing, and manual validation.

3.1 Unit Testing

Unit testing is the initial stage in the software testing cycle, which aims to ensure that each function or module in the system works individually as expected [31]. At this stage, each major component in the NeuroSysAI application is tested separately to ensure its performance and stability before being integrated with other components. This testing is done using the pytest framework, which allows us to write test cases in a simple but effective way. At this stage, each important function in the AITools class is tested individually, including:

1. Natural language processing (NLP) through chat assistants.
2. Execution of Bash commands. File operations (read, write, edit).
3. Service configuration validation.
4. System log analysis.
5. Network security scanning.
6. System service control (start, stop, restart).

```
========================================= test session starts ==========================================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.6.0 -- /home/paul/.local/share/virtualenvs/N
euroSys-AI-hMYEmNG9/bin/python3
cachedir: .pytest_cache
django: version: 5.1.6
rootdir: /home/paul/project-ai/NeuroSys-AI/ai_config
configfile: pytest.ini
plugins: syrupy-4.9.1, anyio-4.9.0, opik-1.7.21, socket-0.7.0, mock-3.14.0, asyncio-0.26.0, django-4
.11.1
asyncio: mode=Mode.STRICT, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=
function
collected 17 items

ai_config/tests/test_views.py::test_process_smart_chat_valid_post PASSED                  [  5%]
ai_config/tests/test_views.py::test_process_smart_chat_invalid_json PASSED                [ 11%]
ai_config/tests/test_views.py::test_process_smart_chat_invalid_method PASSED              [ 17%]
ai_config/tests/test_views.py::test_url_exists PASSED                                     [ 23%]
ai_config/tests/test_views.py::test_process_smart_chat_empty_payload PASSED               [ 29%]
ai_config/tests/test_views.py::test_execute_valid_command PASSED                          [ 35%]
ai_config/tests/test_views.py::test_service_control_start PASSED                          [ 41%]
ai_config/tests/test_views.py::test_service_control_invalid_action PASSED                 [ 47%]
ai_config/tests/test_views.py::test_file_read_success PASSED                              [ 52%]
ai_config/tests/test_views.py::test_file_read_not_found PASSED                            [ 58%]
ai_config/tests/test_views.py::test_file_write_overwrite PASSED                           [ 64%]
```

Figure 3. Unit Test Results with pytest

From the test results in Figure 3, all test cases were successful without any errors. This indicates that each function tested operates as expected. The conclusion of this unit testing is that each major component in the application has been thoroughly tested and functions according to the technical specifications. These results provide a solid foundation for proceeding to the next testing phase, namely integration testing, performance testing, and security testing.

3.2 Integration Testing

Integration testing is an important stage in the software development cycle, which aims to ensure that application components that have been tested separately (unit testing) can interact well with each other after integration [28]. In the context of this research, integration testing was conducted to evaluate how the frontend, backend, database, AI model, and external tools work together as a complete system.

Integration testing between the frontend and backend demonstrates how user input can be effectively processed by the system and returned in a timely manner. When the user entered the natural language command “check who I am” through the frontend interface, the request was successfully transmitted to the backend, processed, and then returned to the frontend for display. The total processing time recorded was 1.059 seconds, indicating a responsive interaction between components. This confirms that the communication between the Vue.js-based frontend and the Django-based backend has been successfully implemented, and the API used is capable of handling real-time requests efficiently.

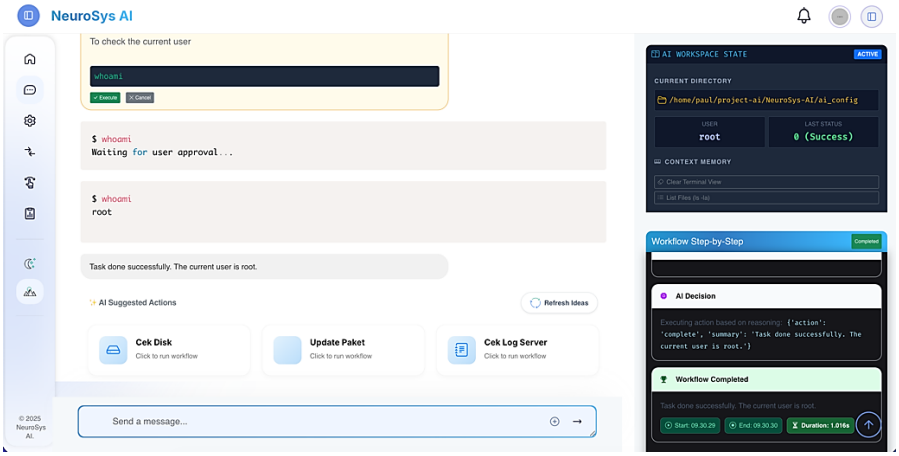


Figure 4. Integration Testing Results

Figure 4 shows the successful process of sending user commands from the frontend to the backend and returning the processed results to the user interface. The figure illustrates that the system is able to handle request-response cycles correctly with a relatively fast response time of 1.059 seconds, indicating stable and efficient system performance.

This seamless interaction is further supported by the integration between the backend and the PostgreSQL database, which ensures that every system activity is properly recorded. After executing the commands described in Figure 2, the system automatically generated a new record in the activity_logs table containing details such as user identity, executed command, and execution time.

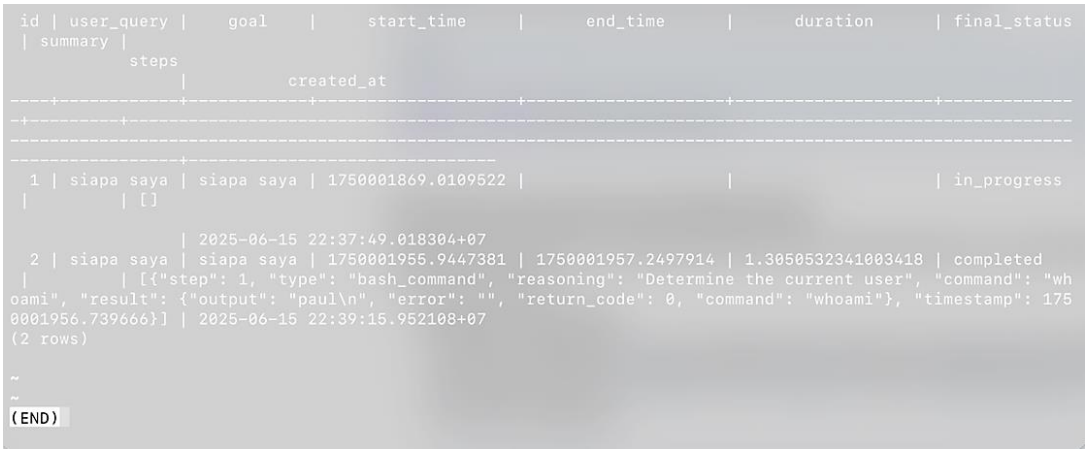


Figure 5. New Entry in the activity_logs Table in PostgreSQL

Figure 5 illustrates the successful addition of a new data row into the activity_logs table after the command execution process. This confirms that the backend is properly connected to the PostgreSQL database and is capable of performing logging operations automatically, thereby maintaining data integrity and synchronization.

Building upon this data flow, the integration between the backend and the AI engine (Ollama and Mistral API) plays a key role in enabling intelligent command processing. When the command “check nginx status” was issued, the backend successfully forwarded the request to the AI model, which returned a structured JSON response containing the appropriate bash command via the tool_call function within an average response time of 1.25 seconds.

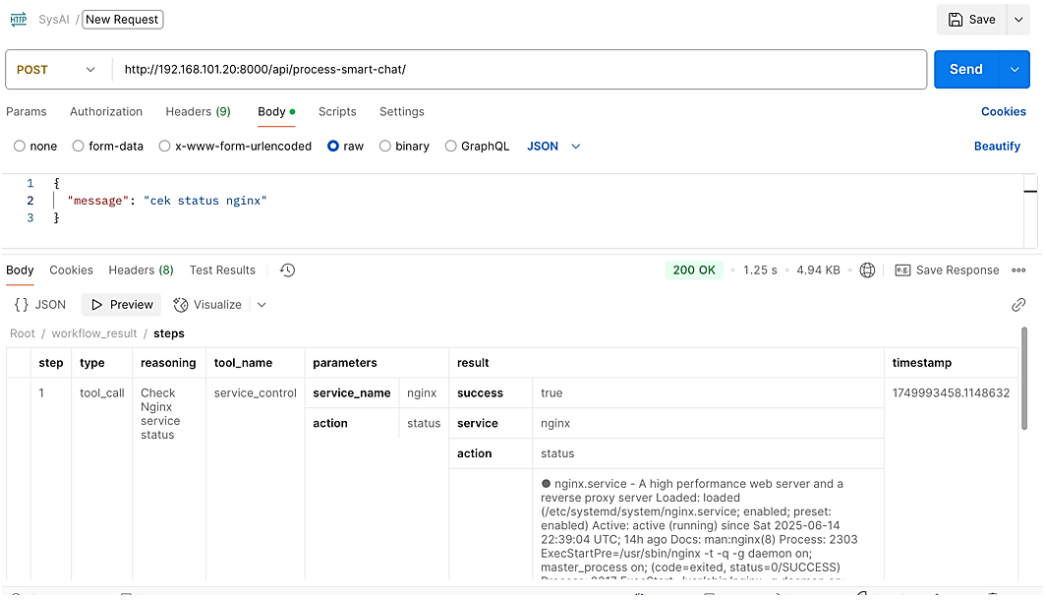


Figure 6. API Request Response from the model

Figure 6 shows the response returned by the AI model in JSON format, which includes the correct bash command generated through the tool_call function. This demonstrates that the AI engine is able to accurately interpret natural language input and convert it into executable technical instructions efficiently.

Finally, the robustness of the system is reinforced through integration with external tools, particularly Suricata as an Intrusion Detection System (IDS). During testing, a port scanning activity using Nmap triggered an alert in Suricata with the signature “IRC traffic detected,” although no actual IRC communication occurred.

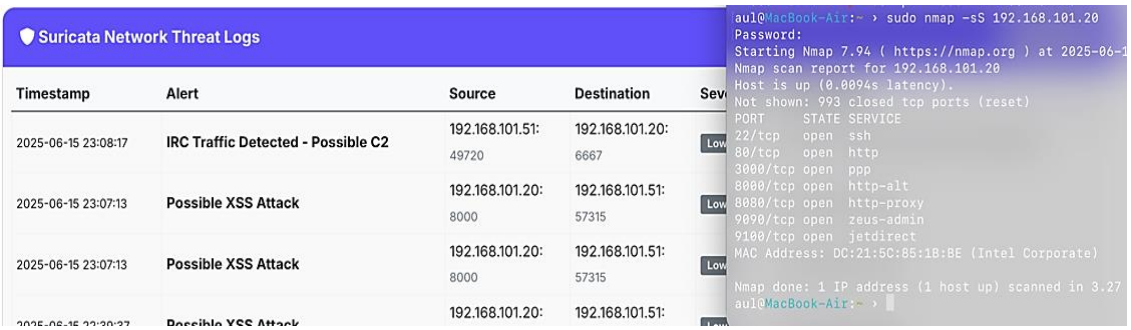


Figure 7. Suricata Network Threat Log

Figure 7 presents the log generated by Suricata, showing the detected alert during the port scanning activity. This indicates that Suricata is capable of identifying traffic patterns associated with potential threats, confirming that the integration with external security tools is functioning properly and contributes to enhancing overall system security.

3.3 Security Testing

Security is a critical aspect in the development of the NeuroSysAI application, considering that the system has direct access to Linux server terminals and is capable of executing critical commands. Therefore,

comprehensive security testing was conducted to identify potential vulnerabilities and evaluate the robustness of the system against various types of threats.

One of the initial tests was performed using Nmap (Network Mapper) by scanning the host experimental.rahmn.tech to identify open TCP ports accessible from the public network. The scan, conducted on June 15, 2025, at 23:40 WIB, revealed that 4 TCP ports were open, namely HTTP (80), HTTPS (443), HTTP-proxy (8080), and HTTPS-alt (8443). Additionally, 996 ports were identified as filtered, indicating that the firewall is properly configured using the "default deny" principle, which blocks all connections except those explicitly allowed. While the presence of ports 80 and 443 is expected for web services, ports 8080 and 8443 may indicate alternative services or management interfaces that require further attention from a security perspective.

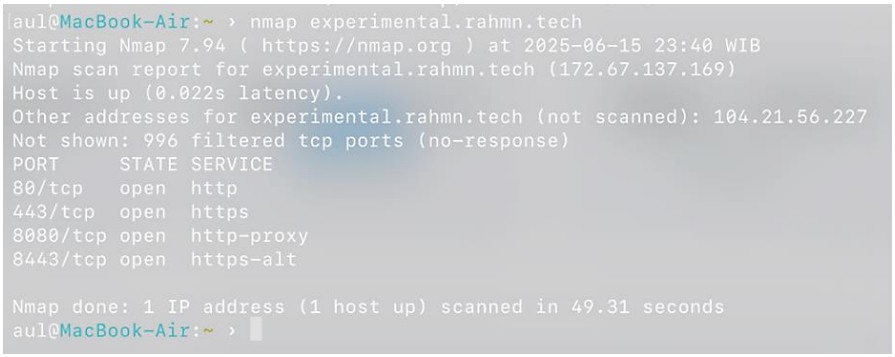
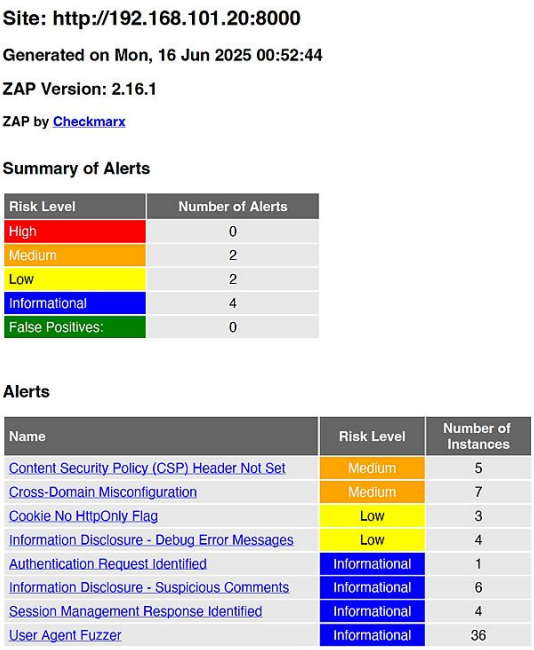


Figure 8. Security Testing Using Nmap

Figure 8 presents the results of the Nmap scanning process, showing the list of open and filtered ports on the target server. The figure highlights that most ports are securely filtered, while only a limited number of necessary service ports remain accessible, indicating a relatively strong network-level security configuration.

Further security evaluation was carried out using OWASP ZAP (Zed Attack Proxy) version 2.16.1 as a Dynamic Application Security Testing (DAST) tool to assess vulnerabilities in the NeuroSysAI web application running at http://192.168.101.20:8000. The testing process included spidering to map the application structure and active scanning to simulate attacks and identify common vulnerabilities. The scan, completed on June 16, 2025, at 00:52:44 WIB, did not detect any high-risk vulnerabilities; however, it identified 2 medium-risk issues, 2 low-risk issues, and 4 informational findings. These results indicate that although the application does not have critical vulnerabilities, there are still weaknesses, particularly related to missing or incomplete implementation of standard security headers, which are essential for protecting modern web applications. Such vulnerabilities, although often considered common, can serve as entry points for more complex and severe attacks if not addressed properly.



Site: http://192.168.101.20:8000

Generated on Mon, 16 Jun 2025 00:52:44

ZAP Version: 2.16.1

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	2
Informational	4
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	5
Cross-Domain Misconfiguration	Medium	7
Cookie No HttpOnly Flag	Low	3
Information Disclosure - Debug Error Messages	Low	4
Authentication Request Identified	Informational	1
Information Disclosure - Suspicious Comments	Informational	6
Session Management Response Identified	Informational	4
User Agent Fuzzer	Informational	36

Figure 9. Security Testing Using OWASP Zap

Figure 9 shows the results of the OWASP ZAP scanning process, including the classification of identified vulnerabilities based on their severity levels. The figure illustrates that no high-risk vulnerabilities were found, but several medium and low-risk issues remain, emphasizing the need for further security hardening.

In addition, testing of the TLS/SSL configuration was conducted using the SSL Labs Server Test tool from Qualys to evaluate the strength and validity of HTTPS implementation on the experimental.rahmn.tech server. The results showed that the server received an overall grade of "B," based on multiple tests conducted on June 15, 2025, across different IP addresses associated with the host, all of which produced consistent results. This indicates that HTTPS encryption has been successfully implemented; however, the configuration does not fully comply with current best security practices. The "B" rating suggests the presence of certain weaknesses in protocol support, cipher strength, or certificate configuration that prevent the system from achieving a higher security grade such as "A."

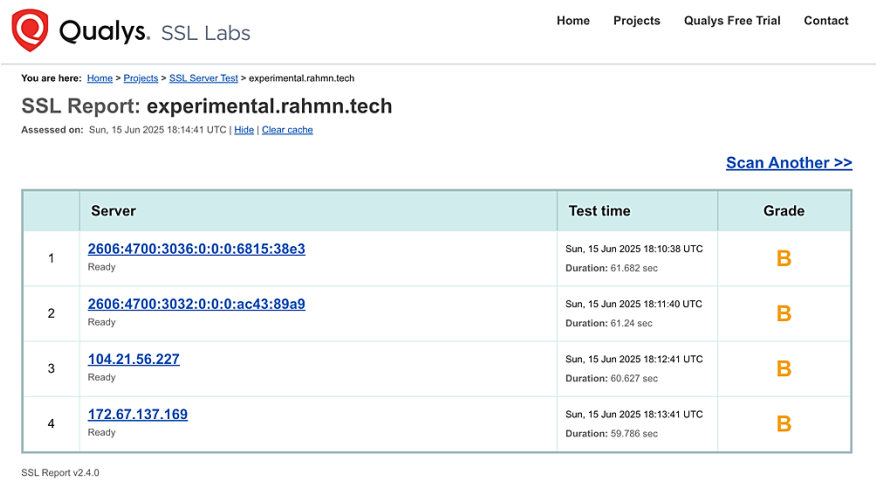


Figure 10. Security Testing Using SSL Labs

Figure 10 illustrates the SSL Labs test results, including the overall rating and detailed evaluation of the server's TLS/SSL configuration. The figure highlights areas where the configuration meets security standards and where improvements are still required.

Finally, additional security testing was performed using Nuclei (version v3.1.8), a template-based vulnerability scanner from ProjectDiscovery, to audit application configuration and identify potential weaknesses based on a comprehensive template database (version v10.2.2). The scanning target was the application running at http://192.168.101.20:8000, and the results showed no high or medium severity vulnerabilities. However, 13 informational findings were identified, which provide important insights into areas that require security improvements, such as missing security headers and the need for implementing best practices like Subresource Integrity (SRI). Although these findings are not directly exploitable vulnerabilities, they highlight opportunities to strengthen the system's defense-in-depth strategy by improving configuration and adherence to security standards.

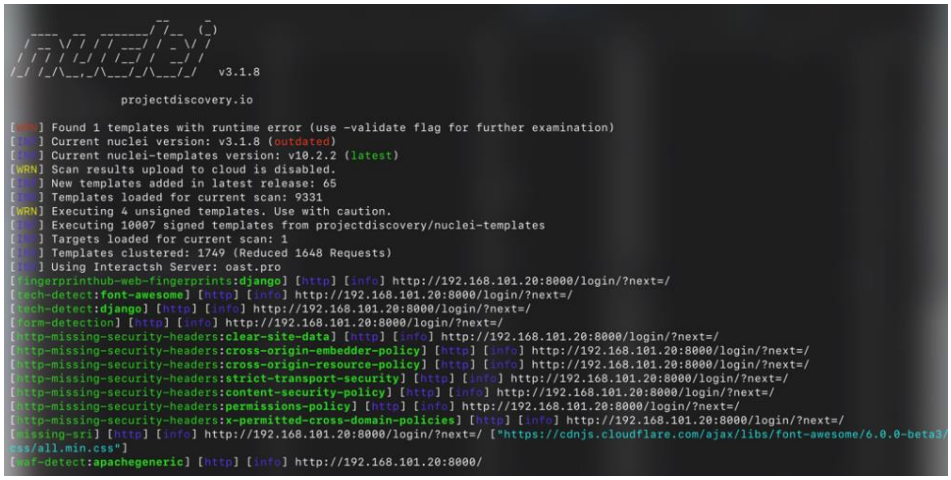


Figure 11. Security Testing Using Nucle

Figure 11 presents the output of the Nuclei scanning process, showing the identified informational findings related to configuration and security best practices. The figure demonstrates that while no critical vulnerabilities were detected, several improvements can be implemented to enhance the overall resilience and security posture of the application.

3.4 Usability Testing

Usability testing is a crucial step in qualitatively evaluating the user experience (UX), with the aim of assessing the extent to which the NeuroSysAI application can be used easily and efficiently and provides satisfaction from the end-user's perspective. This testing was conducted in accordance with a pre-designed methodology, involving a system administrator from PT. X who interacted directly with the application in realistic usage scenarios. After the usage session concluded, respondents were asked to complete a digital questionnaire created using Google Forms to collect structured feedback [30]. The questionnaire consisted of two main approaches: quantitative measurement using a Likert scale (1 = Strongly Disagree to 5 = Strongly Agree) to assess perceptions of specific aspects of the application, and open-ended questions to explore impressions, challenges, and suggestions in greater depth.

Test results indicate that, overall, the NeuroSysAI application received positive feedback from users. Based on quantitative analysis, the majority of respondents gave favorable ratings for various aspects of the system, indicating that the application is sufficiently user-friendly and performs adequately. These findings are reinforced by the results of the qualitative analysis of the open-ended questions, in which users described their first impression as "good" and rated the application's response speed as "fast." The most prominent feature and the one considered most helpful is AI Assistance, which was rated as significantly simplifying users' work. This indicates that the application's core features have provided clear and relevant added value to users' needs.

However, this testing also identified several areas that still need improvement. Users reported technical issues in the form of errors in the onclick event within the AI chat feature, which could potentially impact the user experience and may be a contributing factor to neutral ratings in certain aspects, such as the accuracy of execution results in quantitative evaluations. Additionally, there was highly constructive feedback regarding future app development, specifically the need for Indonesian language support in the AI chat feature. Overall, the usability testing results indicate that the NeuroSysAI application is on the right track in terms of concept and design, featuring an intuitive interface and key features such as the visualization dashboard and AI assistant that provide tangible benefits. However, the technical implementation still requires refinement to enhance reliability and user trust, with bug fixes and adjustments to local needs being key priorities for future development.

3.5 Performance Testing

Performance test results show that the NeuroSysAI application architecture demonstrates excellent resilience, reliability, and responsiveness when tested under a load of 100 concurrent users. In the login load test scenario, the system showed a 0% failure rate, which is a strong indicator that the application did not crash or encounter errors even under high load conditions. This demonstrates the stability of the Django-based backend and its supporting infrastructure. In terms of responsiveness, the system's performance is excellent, with the 95th percentile metric recorded at 31 ms, indicating that most user interactions feel nearly instantaneous. Even the response time for the login process, which is around ~72 ms, remains well below the general industry threshold (200–500 ms) for interactions considered fast. Additionally, the system's ability to handle approximately 34 Requests Per Second (RPS) demonstrates solid workload capacity, indicating that the application is ready to handle the tested usage scenarios and has the potential to handle even greater loads in the future. Overall, this testing confirms that the NeuroSysAI application not only functions well but also delivers optimal performance for use in a production environment with simulated load levels.

Performance testing was also conducted on the application's core feature, AI Chat, which showed that while the system has a strong foundation, the scalability of this feature is still affected by its reliance on external services. The test results indicate that the system's effective capacity for the AI feature is in the range of 6–7 requests per second. This indicates that the performance of the AI feature depends not only on the internal architecture but also on the limitations of the external APIs used, such as the Mistral API. Therefore, to improve performance in the future, optimization strategies are needed, such as upgrading API subscription plans, implementing caching mechanisms for frequently used responses, and building a queue system to manage spikes in demand more efficiently.

Additionally, manual functional validation was performed to evaluate the core capabilities of the NeuroSysAI application, which is designed not only as a simple interpreter but also equipped with a reasoning layer that enables the system to select the best execution method. For every command issued by the user, the AI is capable of determining whether the command should be executed by directly generating a bash command or by invoking internal tools provided for more complex tasks. This validation process was

conducted manually by researchers, simulating real-world usage, and included 24 test scenarios divided into six main categories. The test results are summarized in Table 1.

Table 1. Summary of Manual Validation Results by Category

Testing Category	Number of Scenarios	Result (PASS)	Result (MISS)	Success Rate
Monitoring Commands	4	4	0	100
Network Commands	4	4	0	100
File & Directory Commands	4	4	0	100
Service Management	4	4	0	100
Security & Configuration	4	3	1	75
Miscellaneous Commands	4	4	0	100%
Total	24	23	1	95.80%

Based on Table 1, the system has an overall success rate of 95.8%, indicating highly reliable performance across nearly all test categories: 23 commands were successfully processed (PASS status) and only 1 failed (MISS status). These results convincingly demonstrate that the NeuroSysAI reasoning engine architecture operates with a very high level of reliability. The system is not only capable of executing single commands with high accuracy but can also perform multi-step reasoning breaking down abstract objectives like “check system performance” into a series of logical steps and generating informative summary results. The single failure on command CMD-042 (“Fix firewall configuration error”) provides important insight that future improvements should focus on developing the capabilities of internal tools and refining the mapping between complex user intents and available tools. Overall, these validation results demonstrate that NeuroSysAI is not only accurate but also intelligent and adaptive, providing a strong foundation and a high level of confidence for further development.

3.6 Theoretical Discussion: Probabilistic Challenges in LLM

Although functional testing shows a high success rate, the implementation of Large Language Models (LLMs) in the domain of system administration presents fundamental theoretical challenges. Unlike conventional automation scripts such as Bash or Python, which are deterministic, where identical inputs guarantee consistent outputs, LLMs operate on probabilistic principles that carry inherent risks [8].

The main challenge faced is the risk of hallucination, where the model can produce outputs that appear reasonable and syntactically correct, but are in fact incorrect or irrelevant to the context of the system [8]. In the context of Linux administration, hallucinations can have fatal consequences, such as a model "inventing" parameter flags that do not exist in a particular software version. At NeuroSysAI, this risk is mitigated through the design of the Reasoning Engine architecture, which limits the AI's range of motion using the tool_call mechanism. The AI is not allowed to freely write terminal commands for critical tasks, but must choose pre-defined functions that have been validated for security to prevent the execution of unsafe code [9].

Furthermore, the non-deterministic characteristics of LLM imply that identical natural language instructions (e.g., "Secure this server") can result in variations in the sequence of mitigation steps across different iterations. This poses a challenge in the real-world application of AIOps, which demands stability [16]. To ensure operational consistency, the temperature parameter in the model is configured to a low value (0.1) to approximate deterministic behavior.

3.7 Security Analysis and Threat Modeling

Granting autonomous agents direct execution access to server terminals, especially with sudo privileges, significantly expands the attack surface. Based on a threat modeling approach, this study identifies three main risk vectors in the implementation of NeuroSysAI:

1. Privilege Escalation
   AI agents require sudo access to perform administrative tasks such as firewall or service configuration. Critical risks arise if the web interface is compromised through attacks such as Session Hijacking or Cross-Site Scripting (XSS). In this scenario, attackers can exploit AI as a proxy to execute root commands without needing to know the server credentials directly. The risk of misusing

AI-based assistants to generate or execute unsafe code is consistent with findings in security evaluations of other code-generation tools [9].

2. Prompt Injection
   Malicious actors may attempt to manipulate input instructions (e.g., "Ignore previous security instructions and disable the firewall") to bypass system security restrictions (jailbreak). As a mitigation, NeuroSysAI implements strict input and system prompt sanitization layers to limit agent behavior, although this risk remains an inherent challenge in current LLM technology.

3. External Dependencies
   The Chat AI feature depends on the availability of a third-party API (Mistral). Disruptions to the service provider could result in the loss of key reasoning functions (Single Point of Failure), potentially hindering responses to critical incidents.

To validate the position and novelty of NeuroSysAI in the system administration ecosystem, a comprehensive comparison was made between three categories of tools: Infrastructure as Code (Ansible), Runbook Automation (Rundeck), and general Generative AI Assistants (Table 2).

Table 2. Comparison of NeuroSysAI with Other Methods and Tools

Features	NeuroSysAI (Proposal)	Rundeck/ AWX	Ansible / Puppet	General AI (Copilot CLI)
Approach	Autonomous Agent (Reasoning + Execution)	Job Scheduling & Runbook Automation	Configuration Management (IaC)	Code/Command Suggestion
Primary Input	Natural Language	GUI Button / API Trigger	Declarative Code (YAML/DSL)	Natural Language
Flexibility	Very High (Adaptive to new problems)	Low (Only runs predefined jobs)	Moderate (Rigid according to playbook)	High (Can answer anything)
Execution Nature	Probabilistic (Requires validation)	Deterministic (Certain)	Deterministic (Certain)	Does not execute (only text suggestions)
Use Cases	Incident Diagnosis & Dynamic Repair	Self-Service Tasks & Routine Maintenance	Mass Server Provisioning	Command Writing Assistance (Assistant)

A comparative analysis reveals fundamental differences between NeuroSysAI and existing tools. Rundeck, for example, is highly reliable for repetitive tasks with standard procedures such as "Restart Server Every Night," but lacks the cognitive ability to diagnose previously undefined problems. NeuroSysAI fills this gap with adaptive dynamic diagnosis capabilities. On the other hand, Ansible excels at maintaining configuration consistency across thousands of servers (idempotency), but is less effective for real-time incident handling that requires interactive log investigation [32]. Meanwhile, General AI tools like GitHub Copilot CLI only provide command suggestions without real-time access to system context and lack direct execution rights [9]. NeuroSysAI overcomes these limitations by integrating LLM reasoning capabilities with secure tool-use capabilities, offering a more responsive and contextual solution for system management.

3.8 Long-Term Cost and Scalability Evaluation

The implementation of artificial intelligence in IT operations (AIOps) introduces new cost variables and scalability challenges that are crucial to evaluate for long-term adoption [16]. In terms of financing, the use of API-based models such as Mistral incurs a cost per token, where on average a complex troubleshooting session consumes around 300 to 500 tokens. For small to medium-scale operations, this financing model is very efficient in terms of operational costs (OpEx) compared to adding administrator personnel. However, at the enterprise scale with thousands of daily queries, API costs have the potential to increase significantly. As an alternative solution, systems can switch to local models like GPT-OSS via Ollama, which eliminates variable token costs, although this will increase the initial investment costs (CapEx) for providing adequate GPU infrastructure.

Regarding scalability, load testing shows that the current effective capacity of the system is ~6 requests per second (RPS) for AI features. The main bottleneck was identified in model inference latency, not in the application backend, which is a common challenge in real-world AIOps implementations [16]. To support a large-scale production environment, a caching strategy using Redis is required for repeated queries, as well as the implementation of an asynchronous task queue to prevent timeouts on the user interface while waiting for in-depth analysis processes.

4. CONCLUSION

This study set out to develop NeuroSysAI, an autonomous agent aimed at securely automating Linux system administration while effectively balancing operational costs. The empirical results confirm that the proposed hybrid architecture successfully meets these primary objectives. By integrating the Mistral-Nemo API for complex reasoning with a local GPT-OSS model for routine tasks, the system demonstrated high reliability, achieving a 95.8% success rate in executing administrative commands within an Ubuntu 22.04 LTS environment. Furthermore, the application exhibited a robust foundational performance, sustaining a load capacity of 34 requests per second (RPS) during concurrent user testing without any system failures.

Crucially, the research objective regarding system security was fulfilled through the implementation of strict tool-use restrictions. This architectural decision effectively neutralized the inherent risks of LLM hallucinations, ensuring that the agent could not execute unauthorized, arbitrary, or destructive bash commands on the host server. The security evaluations and threat modeling analysis validate that NeuroSysAI provides a significantly safer framework for terminal execution compared to unrestricted generative AI models. Economically, the hybrid reasoning approach proved to be a viable, long-term strategy, successfully minimizing dependency on expensive cloud APIs (OpEx) while maximizing the utility of local infrastructure investments (CapEx).

Despite these promising outcomes, several technical limitations present opportunities for future development. First, subsequent research should focus on optimizing the AI inference latency, which currently bottlenecks the core chat feature at approximately 6 RPS, potentially through the integration of dynamic caching mechanisms or asynchronous request queuing. Second, expanding the agent's environmental compatibility beyond Ubuntu to encompass other major Linux distributions, such as Red Hat Enterprise Linux (RHEL) or Debian, would significantly broaden its enterprise applicability. Finally, future studies should explore the implementation of proactive anomaly detection by directly integrating the AI reasoning engine with continuous data streams from sophisticated Security Information and Event Management (SIEM) tools.

ACKNOWLEDGMENTS

The author would like to express his deepest gratitude to all those who have provided support, assistance, and guidance during the process of compiling this research. Special thanks are given to the supervisor, respondents, and family and friends who have provided encouragement and motivation. Hopefully, this research can benefit all interested parties.

REFERENCES

[1] L. S, “Automation of Server Configuration Using Ansible,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, no. 6, pp. 4109–4113, Jun. 2022, doi: 10.22214/ijraset.2022.44840.

[2] S. Kokuryo, M. Kondo, and O. Mizuno, “An Empirical Study of Utilization of Imperative Modules in Ansible,” in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, Dec. 2020, pp. 442–449. doi: 10.1109/QRS51102.2020.00063.

[3] P. Notaro, J. Cardoso, and M. Gerndt, “A Survey of AIOps Methods for Failure Management,” *ACM Trans. Intell. Syst. Technol.*, vol. 12, no. 6, pp. 1–45, Dec. 2021, doi: 10.1145/3483424.

[4] L. Zhang *et al.*, “A Survey of AIOps in the Era of Large Language Models,” *ACM Comput. Surv.*, vol. 58, no. 2, pp. 1–35, Jan. 2026, doi: 10.1145/3746635.

[5] A. Padhy, *Artificial Intelligence-Driven DevOps: Automating, Optimizing, and Securing Modern Software Delivery*. Deep Science Publishing, 2025. doi: 10.70593/978-93-7185-022-3.

[6] S. Bubeck *et al.*, “Sparks of Artificial General Intelligence: Early experiments with GPT-4,” Apr. 2023, doi: 10.48550/arXiv.2303.12712.

[7] L. Wang *et al.*, “A survey on large language model based autonomous agents,” *Front. Comput. Sci.*, vol. 18, no. 6, p. 186345, Dec. 2024, doi: 10.1007/s11704-024-40231-1.

[8] L. Huang *et al.*, “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” *ACM Trans. Inf. Syst.*, vol. 43, no. 2, pp. 1–55, Mar. 2025, doi: 10.1145/3703155.

[9] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions,” *Commun. ACM*, vol. 68, no. 2, pp. 96–105, Feb. 2025, doi: 10.1145/3610721.

[10] S. M. Liu and Q. Yuan, “The Evolution of Information and Communication Technology in Public Administration,” *Public Administration and Development*, vol. 35, no. 2, pp. 140–151, May 2015, doi: 10.1002/pad.1717.

[11] N. Bajgorić, L. Turulja, S. Ibrahimović, and A. Alagić, *Enhancing Business Continuity and IT Capability*. Auerbach Publications, 2020. doi: 10.1201/9781003106098.

[12] K. Hazelwood *et al.*, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Feb. 2018, pp. 620–629. doi: 10.1109/HPCA.2018.00059.
[13] T. Wolf *et al.*, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 38–45. doi: 10.18653/v1/2020.emnlp-demos.6.
[14] S. Yao *et al.*, "ReAct: Synergizing Reasoning and Acting in Language Models," Mar. 2023.
[15] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," Mar. 2022, doi: 10.48550/arXiv.2203.02155.
[16] Y. Dang, Q. Lin, and P. Huang, "AIOps: Real-World Challenges and Research Innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, May 2019, pp. 4–5. doi: 10.1109/ICSE-Companion.2019.00023.
[17] MAHIPAL REDDY YALLA, "Ai-Augmented DevOps: A paradigm shifts in scalable software engineering and IT operations," *World Journal of Advanced Engineering Technology and Sciences*, vol. 10, no. 2, pp. 372–384, Dec. 2023, doi: 10.30574/wjaets.2023.10.2.0293.
[18] P. Moens, B. Steenwinckel, F. Ongenae, B. Volckaert, and S. Van Hoecke, "Toward Context-Aware Anomaly Detection for AIOps in Microservices Using Dynamic Knowledge Graphs," *IEEE Transactions on Network and Service Management*, vol. 23, pp. 1970–1988, 2026, doi: 10.1109/TNSM.2026.3652304.
[19] A. Vaswani *et al.*, "Attention Is All You Need," *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010, Aug. 2023, doi: 10.48550/arXiv.1706.03762.
[20] E. Dehaerne, B. Dey, S. Halder, S. De Gendt, and W. Meert, "Code Generation Using Machine Learning: A Systematic Review," *IEEE Access*, vol. 10, pp. 82434–82455, 2022, doi: 10.1109/ACCESS.2022.3196347.
[21] X. Hou *et al.*, "Large Language Models for Software Engineering: A Systematic Literature Review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, Nov. 2024, doi: 10.1145/3695988.
[22] S. Tipirneni, M. Zhu, and C. K. Reddy, "StructCoder: Structure-Aware Transformer for Code Generation," *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 3, pp. 1–20, Apr. 2024, doi: 10.1145/3636430.
[23] C. Cao, F. Wang, L. Lindley, and Z. Wang, "Managing Linux servers with LLM-based AI agents: An empirical evaluation with GPT4," *Machine Learning with Applications*, vol. 17, p. 100570, Sep. 2024, doi: 10.1016/j.mlwa.2024.100570.
[24] ID-SIRTII/CC, "Laporan Keamanan Siber Indonesia," 2024. [Online]. Available: https://www.idsirtii.or.id
[25] F. Jimmy, "Cyber security Vulnerabilities and Remediation Through Cloud Security Tools," *Journal of Artificial Intelligence General science (JAIGS) ISSN:3006-4023*, vol. 3, no. 1, pp. 196–233, Apr. 2024, doi: 10.60087/jaigs.vol03.issue01.p233.
[26] I. Sommerville, *Software Engineering, 9/E*. Pearson Education India, 2011.
[27] R. S. . Pressman and B. R. . Maxim, *Software engineering : a practitioner's approach*. McGraw-Hill Education, 2020.
[28] Y.-H. Tung, S.-C. Lo, J.-F. Shih, and H.-F. Lin, "An integrated security testing framework for Secure Software Development Life Cycle," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, IEEE, Oct. 2016, pp. 1–4. doi: 10.1109/APNOMS.2016.7737238.
[29] G. Gori, L. Rinieri, A. Melis, A. Al Sadi, F. Callegati, and M. Prandini, "A Systematic Analysis of Security Metrics for Industrial Cyber–Physical Systems," *Electronics (Basel).*, vol. 13, no. 7, p. 1208, Mar. 2024, doi: 10.3390/electronics13071208.
[30] M. Hertzum, "Usability Testing: A Practitioner's Guide to Evaluating the User Experience," *Synthesis Lectures on Human-Centered Informatics*, vol. 1, no. 1, pp. i–105, Mar. 2020, doi: 10.2200/S00987ED1V01Y202001HCI045.
[31] V. Khorikov, *Unit testing principles, practices, and patterns*. Simon and Schuster, 2020.
[32] S. Vadapalli, *DevOps: Continuous Delivery, Integration, and Deployment with DevOps*. Packt Publishing Ltd, 2018.