Intelligent system for automatic detection of web application vulnerabilities and threat classification*

Andrii Nicheporuk[1,*,†], Andrii Selskyi[1,†], Andrii Ramskyi[1,†], Jiří Balej[2,†], Dmytro Mykuliak[1,†], Tomas Sochor[2,3,†]

[1] *Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, 29016, Ukraine*
[2] *Mendel University in Brno, 1665/1, Zemědělská, Brno, 613 00, Czech Republic*
[3] *European Research University, Ostrava, Czech Republic*

Abstract

The transition from monolithic to microservice-based systems, the widespread use of third-party components, and the acceleration of release cycles under CI/CD and DevSecOps paradigms have made it practically impossible to rely on periodic manual security audits. This paper presents an Intelligent System for Automatic Vulnerability Detection and threat classification in web applications (ISAVDWA). The core contribution is a hybrid neural architecture that combines Graph Neural Networks (GNN) and Large Language Models (LLM) within a unified analytical pipeline. GNNs capture structural properties of source code by modeling it as a directed data-flow and control-flow graph, enabling the system to trace how potentially dangerous values propagate across modules and functions. LLMs complement this with deep semantic understanding, allowing the detection of logical vulnerabilities such as broken access control or business logic flaws that remain invisible to classical pattern-matching static analyzers. A key architectural element is the normalization module, which aggregates heterogeneous outputs from three complementary scanning techniques – Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Software Composition Analysis (SCA) into a unified feature space. This resolves the fragmentation problem inherent in conventional pipelines, where each tool generates reports in its own format (JSON, XML, SARIF) without a shared representation. When a static analyzer flags a suspicious code pattern and a dynamic scanner independently confirms its exploitability via an HTTP request, the normalization operator merges both signals into a single enriched record, improving accuracy while reducing noise.

Keywords

web application vulnerabilities, threats, Static Application Security Testing, Dynamic Application Security Testing, Software Composition Analysis [1]

1. Introduction

The current stage of information technology development is inextricably linked with the global digitalization of business processes and the migration of critical infrastructure to the online environment. Web applications and cloud architectures form the foundation of this digitalization. At the same time, this stage is characterized by the rapid complication of software architecture. The transition from monolithic applications to microservice architectures, the integration of a vast number of third-party components, and the use of containerization and API-oriented approaches have significantly expanded the surface for potential cyberattacks. Consequently, web application security has become one of the industry's most critical challenges. Despite the existence of a developed international regulatory framework, including the ISO/IEC 27001 and ISO/IEC 27002 standards for implementing information security management systems and ISO/IEC 27005 in the

* *IntelITSIS-2026: 7th International Workshop on Intelligent Information Technologies & Systems of Information Security, July 3, 2026, Khmelnytskyi, Ukraine*
[1,*] Corresponding author.
[†] These authors contributed equally.
andrey.nicheporuk@gmail.com (A. Nicheporuk); andriy.saa@gmail.com (A. Selskyi); ramskyiao@khmnu.edu.ua (A. Ramskyi); jiri.balej@mendelu.cz (J. Balej); mykuliak.dmytro.ua@gmail.com (D. Mykuliak); tomas.sochor@osu.cz (T. Sochor)
0000-0002-7230-9475 (A. Nicheporuk); 0000-0002-7373-0472 (A. Selskyi); 0000-0001-9624-5018 (A. Ramskyi); 0000-0002-6054-8700 (J. Balej); 0009-0000-9437-8685 (D. Mykuliak); 0000-0002-1704-1883 (T. Sochor)

context of risk management, as well as global vulnerability taxonomies such as Common Vulnerabilities and Exposures (CVE), Common Weakness Enumeration (CWE) [1], and Common Attack Pattern Enumeration and Classification (CAPEC) [2], the practical implementation of threat detection, classification, and prioritization processes within real continuous development and deployment pipelines (CI/CD and DevSecOps) remains extremely fragmented [3, 4].

According to reports from international analytical agencies and the Open Worldwide Application Security Project (OWASP) database, over 80% of modern web applications contain at least one critical vulnerability at the time of release. Traditional security approaches based on periodic manual audits or the localized application of isolated verification tools prove ineffective under the Agile concept, where new product versions may be released several times a day. Automated analysis methods, such as Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and Software Composition Analysis (SCA), have become the de facto industry standards [5]. However, their disparate use is unable to provide a holistic vision of the product's security state. An analysis of the current situation in automated security testing highlights several critical problems requiring urgent scientific and engineering solutions [6].

One primary issue is the heterogeneity of analysis results and the lack of unification. Specifically, there is no unified data integration scheme for information obtained from different tools. SAST systems analyze source code to identify structural flaws, such as hardcoded passwords or a lack of input validation. DAST systems interact with the running application through HTTP requests to simulate attacker behavior, while SCA scanners check third-party open-source libraries for known CVEs. Each of these tools generates reports in its own format, such as JSON, XML, or SARIF, and utilizes its own algorithms for determining criticality and its own terminology. This complicates data aggregation, correlation, and deduplication, creating chaos in the process of remediating identified problems.

The problem of high false-positive rates is another significant challenge. Static analyzers, having access to the entire codebase, often utilize rigid regular expressions or simple lexical analyzers [7, 8]. As a result, they generate a vast number of alerts for potential vulnerabilities that do not pose a threat in a real execution context; for instance, a variable that has not undergone sanitization in one module might be cleaned at the middleware level before reaching the database. Conversely, DAST analyzers can produce false negatives due to an inability to access hidden endpoints or complex authentication mechanisms. This constant stream of false alerts leads to alert fatigue, causing security professionals to ignore scanner messages and potentially miss truly critical threats.

Furthermore, the existing paradigm for determining criticality relies mostly on the basic Common Vulnerability Scoring System (CVSS). However, basic CVSS scoring only evaluates the technical complexity of exploiting a vulnerability and its theoretical impact on the confidentiality, integrity, and availability (CIA) triad [9, 10]. This metric is entirely static and fails to account for the application's business context, the architectural specifics of the deployment environment, the presence of compensating mechanisms such as Web Application Firewalls, and the value of the compromised assets. Consequently, a vulnerability with a high base score in an internal testing environment might be viewed by tools as a higher priority for remediation than a medium-score vulnerability on a public payment gateway.

Despite the existence of detailed taxonomies, a semantic gap is observed in practice. There is an absence of an automated, formalized link in the threat formation chain, ranging from the abstract type of developer error classified by CWE, through a specific vulnerability instance represented by CVE, to a potential attack method or pattern defined by CAPEC [11]. Analyzers point to a specific line of code but do not provide an understanding of the exploitation scenario, which is critical for building defense systems such as deception systems or traps. While specialized standards like ISO/IEC 29147 regarding vulnerability disclosure and ISO/IEC 30111 regarding vulnerability handling processes clearly regulate organizational and management aspects, they do not define specific mathematical or algorithmic mechanisms for the intelligent aggregation of analysis results.

In traditional systems, this process is still performed manually, which is unacceptable for large-scale corporate environments [12].

Thus, there is an acute scientific and practical need to create a formalized integrated model that provides the synchronization of taxonomic knowledge for a deep semantic understanding of the nature of threats. Such a model must ensure the adaptive integration of multi-tool scanning results by bringing heterogeneous data into a single feature space and provide dynamic risk scoring that combines base CVSS metrics with contextual exploitation coefficients according to the recommendations of the ISO/IEC 27005 risk management standard. Solving this multi-dimensional problem with classical deterministic algorithms is impossible due to the high variability of source code and the volatility of the cyber threat landscape. One effective path is the development of an intelligent computer system capable of automatic feature extraction, abstraction, and machine learning on large datasets of vulnerabilities. The involvement of deep learning methods, particularly Graph Neural Networks (GNN) for structural code analysis and Large Language Models (LLM) for the contextual processing of natural language and programming syntax, opens new perspectives for creating systems capable of not only detecting but also intelligently classifying and prioritizing cyber threats with an accuracy that approaches human expertise.

2. Related works

The current stage of information technology development is inextricably linked with the global digitalization of business processes and the migration of critical infrastructure to the online environment. Web applications and cloud architectures form the foundation of this digitalization. At the same time, this stage is characterized by the rapid complication of software architecture. The transition from monolithic applications to microservice architectures, the integration of a vast number of third-party components, and the use of containerization and API-oriented approaches have significantly expanded the surface for potential cyberattacks. Consequently, web application security has become one of the industry's most critical challenges. Despite the existence of a developed international regulatory framework, including the ISO/IEC 27001 and ISO/IEC 27002 standards for implementing information security management systems and ISO/IEC 27005 in the context of risk management, as well as global vulnerability taxonomies such as Common Vulnerabilities and Exposures (CVE), Common Weakness Enumeration (CWE), and Common Attack Pattern Enumeration and Classification (CAPEC), the practical implementation of threat detection, classification, and prioritization processes within real continuous development and deployment pipelines (CI/CD and DevSecOps) remains extremely fragmented. According to reports from international analytical agencies and the Open Worldwide Application Security Project (OWASP) database, over 80% of modern web applications contain at least one critical vulnerability at the time of release. Traditional security approaches based on periodic manual audits or the localized application of isolated verification tools prove ineffective under the Agile concept, where new product versions may be released several times a day. Automated analysis methods, such as SAST, DAST, and SCA, have become the de facto industry standards. However, their disparate use is unable to provide a holistic vision of the product's security state.

The automation of vulnerability detection, the reduction of false positive rates, and the intellectualization of cybersecurity systems are at the center of attention for many international research institutions. A detailed analysis of contemporary professional literature allows for the classification of existing approaches along several key vectors. The foundation for any automated detection system lies in standards and knowledge bases maintained by international communities such as the MITRE Corporation and the OWASP Foundation. In work [13], the application of the CWE and CAPEC taxonomies is described in detail, enabling the formalization of code error types and the methods used by attackers to exploit them. However, as the authors of the OWASP methodology [14] correctly point out, while static lists provide a general understanding of trends,

they require constant adaptation to the dynamic microservice environment of modern web applications. Risk management issues and incident criticality assessments traditionally rely on the ISO/IEC 2700x series of standards. These establish the conceptual frameworks for building Information Security Management Systems (ISMS), but as noted in general provisions [15], they leave significant gaps in the engineering and algorithmic implementation of prioritization processes at the software code level.

Modern research demonstrates a rapid shift from classical machine learning methods toward the use of deep learning architectures. Specifically, authors in works [16, 17] conducted a large-scale analysis of applying transformer-based architectures, such as BERT, for malware detection and automated source code auditing. They prove that the ability of transformers to maintain context over large sections of code allows for a significant reduction in the false positive rate (FPR). Furthermore, researchers in works [18, 19] emphasize the extraordinary effectiveness of using Large Language Models (LLMs) for the semantic analysis of source code. The authors argue that LLMs are capable of identifying complex business logic vulnerabilities and broken access control errors, which are conceptually inaccessible to classical deterministic SAST tools based on abstract syntax trees or regular expressions.

A particularly significant direction in the structural analysis of source code is the application of Graph Neural Networks (GNN), which model code as a directed graph of data and control dependencies rather than a flat text sequence. Authors of [20] proposed BGNN4VD — a bidirectional GNN architecture for vulnerability detection that constructs a composite graph from the abstract syntax tree (AST), control flow graph (CFG), and data flow graph (DFG), and then introduces backward edges to capture information that unidirectional propagation misses. Evaluated on four large C/C++ open-source projects from NVD and GitHub, BGNN4VD demonstrated improvements of 4.9%, 11.0%, and 8.4% in F1-measure, accuracy, and precision respectively over prior state-of-the-art detectors. This work established the foundational principle — later adopted in ICSAVD — that bidirectional graph traversal over composite code representations captures richer syntactic and semantic features than any single graph type alone. Complementing this, authors [21] introduced LineVD, which reformulates vulnerability detection as a node classification task within a GNN, enabling localization at the individual statement level rather than the function level. By combining control and data dependency edges with a Transformer-based encoder for raw source code tokens and resolving conflicting signals between function-level and statement-level labels, LineVD achieved a 105% improvement in F1-score over the previous state of the art on real-world C/C++ vulnerability datasets. The significance of LineVD for the present work lies in demonstrating that fine-grained, line-level annotations — analogous to the annotation strategy adopted in ICSAVD's dataset construction — substantially improve model precision and developer utility compared to file- or function-level labeling.

The integration of LLMs into static analysis pipelines represents another rapidly maturing research direction. Authors in [22] presented IRIS — a system that augments the classical static analyzer CodeQL with LLM-driven specification inference and false positive filtering. Using models ranging from compact open-source variants such as Llama 3 8B and DeepSeekCoder 7B to frontier models such as GPT-4, IRIS demonstrates that even locally deployable models can meaningfully reduce the number of spurious alerts while retaining high recall on real vulnerability benchmarks. Crucially, the authors identify a persistent limitation: LLMs struggle with multi-file dependency chains and deep call stacks, precisely the structural patterns that graph-based methods are designed to capture. This complementarity between LLM semantic understanding and GNN structural analysis directly motivates the hybrid architecture of ICSAVD.

The development of intelligent defensive mechanisms also extends to the detection of sophisticated threats and the architectural evaluation of infrastructure. Research in works [23, 24] explores advanced methods for identifying unknown metamorphic viruses through the analysis of obfuscation features and specialized detection algorithms. This structural approach is further

enhanced by searching for equivalent functional blocks to mitigate the effects of code mutation [25]. Simultaneously, the need for high-level assessment is addressed through systems designed for the comprehensive cybersecurity evaluation of corporate networks [26]. To counteract active threats, modern architectures increasingly incorporate deception-based components, such as traps and baits, where the determination of optimal centralization criteria plays a vital role in maintaining system resilience [27]. The broader challenge of managing and coordinating distributed computational resources – relevant to the deployment of multi-component security pipelines — has been addressed through Kubernetes-based orchestration of FPGA devices in distributed systems [28], decentralized blockchain frameworks for provenance and audit trail integrity [29], and formal models of knowledge potential distribution in networked agent communities [30, 31].

Despite the significant successes of the scientific community in individual fields, most existing systems focus fragmentarily on either the improvement of the detection process or exclusively on the categorization process, without providing a closed cycle of intelligent security incident processing. Currently, there is a lack of comprehensive formalized models that would simultaneously normalize heterogeneous data from SAST, DAST, and SCA into a single feature space, utilize the synergy of Graph Neural Networks and LLMs for deep analysis, and provide dynamic, context-oriented risk-scoring that complements standard CVSS assessments. The development of such an intelligent system represents a pressing scientific and applied challenge, to which this study is dedicated.

3. Intelligent system for automatic detection of web application vulnerabilities and threat classification

The operation of the developed Intelligent System for Automatic Vulnerability Detection of Web Applications (ISAVDWA) is based on the principles of multi-level data processing, which allows for the transformation of disparate technical analysis results into a structured decision-making framework. The architecture is built upon a mathematical model where each web application is viewed as a dynamic set of structural and logical artifacts. Formally, this relationship is described through the set $A(w_i)=\{a_{i1}, a_{i2}, \ldots, a_{iK}\}$ where the elements of analysis include not only source code files but also complex dependency graphs, configuration modules, and specific HTTP requests. Such a comprehensive approach enables the system to cover the entire attack surface, ranging from low-level implementation errors to complex logical defects.

The intelligent analysis process begins with the formation of a feature space, where, using the operator Φ, each artifact is mapped into a high-dimensional vector embedding $x=\Phi(a_{ik}), x \in R^d$. This is critical for subsequent classification according to the CWE taxonomy, such as SQL injections (CWE-89) or cross-site scripting (CWE-79). Unlike traditional systems, ISAVDWA addresses the issue of isolation between different scanning types by creating a unified incident space $S_{\text{total}}=S_{\text{SAST}} \cup S_{\text{DAST}} \cup S_{\text{SCA}}$. The implementation of a normalization operator $N: S_{\text{total}} \to R^d$ ensures automatic deduplication and data correlation. For instance, if a static analysis tool (SAST) flags a suspicious code segment and a dynamic scanner (DAST) confirms the possibility of its exploitation via a network request, the system merges these signals into a single vector, significantly increasing the reliability of the findings.

The central component of the system is a hybrid neural network model that combines the structural precision of Graph Neural Networks (GNN) with the semantic depth of Large Language Models (LLM). The graph component analyzes the code as a directed graph $G=(V, E)$, where nodes represent operators and variables, and edges represent data and control flows. Simultaneously, the language component, based on the Transformer architecture, interprets the textual semantics of the code and developer comments. The final decision regarding the presence of a threat is formed as a weighted ensemble of probabilities:

$$P_{\text{final}}(y/x)=\alpha \cdot P_{\text{GNN}}(y/x)+(1-\alpha)\, P_{\text{LLM}}(y/x), \quad (1)$$

where the coefficient α is optimized to achieve maximum accuracy.
This synergy allows for the detection of complex business logic vulnerabilities that are typically ignored by classical deterministic algorithms.

The final and most critical stage of the system's operation is dynamic contextual risk scoring, which overcomes the primary drawback of the standard CVSS—its detachment from real-world exploitation conditions and the business environment. Since the base CVSS score only evaluates theoretical technical complexity, it can be misleading, causing security teams to waste resources on "critical" vulnerabilities in secondary systems. To solve this, ISAVDWA utilizes an integral risk index:

$$\text{Risk}_{\text{total}}=\text{BaseScore}_{\text{CVSS}} \cdot K_{\text{asset}} \cdot K_{\text{exposure}}, \quad (2)$$

The calculation of this index involves the automatic integration of data from both internal and external sources. The $\text{BaseScore}_{\text{CVSS}}$ is obtained directly from scan results or via queries to global CVE databases. The business criticality coefficient K_{asset} is determined based on asset metadata: the system analyzes the module's role in the general architecture, its access to sensitive data, or critical business functions (e.g., payment gateways). The value of K_{asset} ranges from minimum for testing environments to maximum for production systems processing personal data. The exposure coefficient K_{exposure} is calculated based on results of dynamic application security testing and network configurations. If a vulnerability is found on a public API accessible from anywhere, K_{exposure} approaches one; for internal services protected by multi-level authentication or a web application firewall , this coefficient is significantly reduced.

This methodology allows the system to move beyond generating a dry list of technical flaws and instead produce an adaptive threat mitigation plan, prioritizing attack vectors that pose a real danger to business viability. Thus, the Information Processing Flows in an Intelligent System for Web Application Vulnerability Detection and Threat Classification are presented in Fig. 1.

In general, the functioning of the system can be represented as a sequence of the following stages:

1. Data Sources. The system ingests artifacts from the web application itself (source code, requests) alongside results from multi-instrument scanning, including static analysis (SAST), dynamic analysis (DAST), and software composition analysis (SCA),. This approach overcomes the isolation of individual tools, which typically hinders a holistic view of security.
2. Preprocessing and Normalization. At this stage, reports in various formats (such as JSON, XML, or SARIF) are transformed into a unified feature space,. The system performs automatic deduplication and data correlation—for example, confirming a theoretical suspicion from a static analyzer with a practical result from dynamic scanning.
3. Hybrid Neural Network Ensemble. This is the central intelligent node where data is processed in parallel by two different models GNN and LLM. GNN focuses on structural analysis, treating the code as a system of interconnections to analyze data flows and program control logic, while LLM 0erforms semantic analysis of the code text and vulnerability context, enabling the detection of complex logical defects that are usually inaccessible to classical algorithms.
4. Prediction Aggregation: The results from both neural networks are combined through weighted probability averaging. This ensures synergy: the system simultaneously utilizes the structural precision of graph analysis and the deep contextual understanding of the language model.
5. Output Results and Dynamic Risk Scoring. The system produces more than just a list of errors; it provides classified threats with a calculated real-world risk level,. Risk is

calculated as a comprehensive index that considers the technical severity of the vulnerability, the business criticality of the specific module, and its actual exposure to attacks from external networks.

6. Feedback and Training. The model includes a stage for retraining based on confirmed incidents. This ensures continuous improvement of the system's accuracy and helps significantly reduce the false positive rate over time.

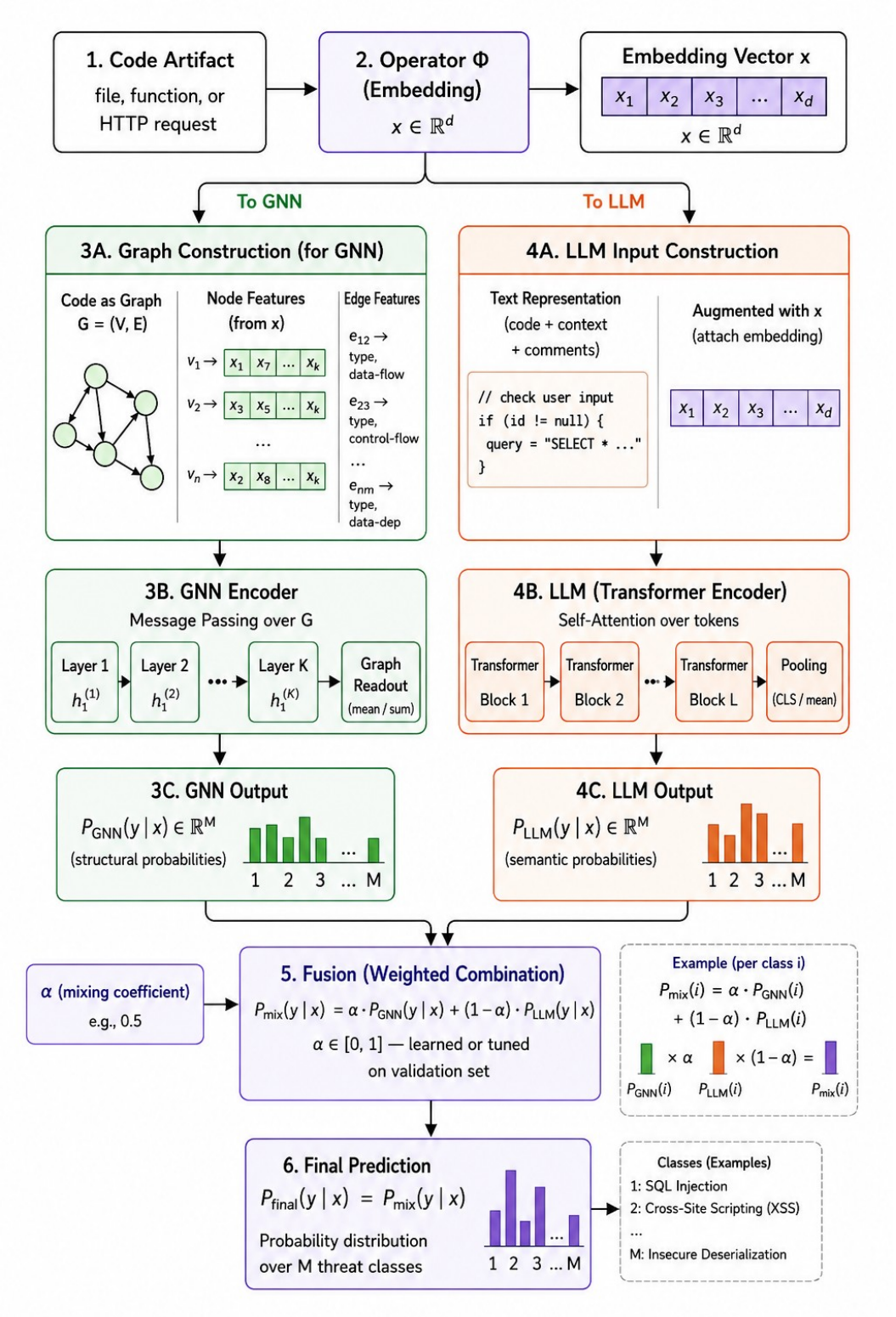

Figure 1: Information Processing Flows in an Intelligent System for Web Application Vulnerability Detection and Threat Classification.

4. Evaluation

The goal of the experimental part is to empirically verify whether the hybrid GNN + LLM architecture truly outperforms classical approaches and, if so, how significantly. To this end, all four architectures (Random Forest, GNN, LLM, and their ensemble) were trained and tested on the same dataset under identical conditions. None of the models had privileged access to the test data – the samples were pre-split and "frozen" before training commenced.

This eliminates the possibility of results fitting or data leakage. The comparison encompassed six metrics: Precision, Recall, F1-score, Area Under the ROC Curve (ROC-AUC), False Positive Rate (FPR), and average analysis time per commit.

In our evaluation tests the final sample includes N=14,042 security events and consists of two different groups: real-world vulnerabilities and synthetic data.

Real-world vulnerabilities are include 62.3% or 8,742 records. The data for this group was sourced from 12 open-source web projects written in three languages: Python, Java, and Node.js. The selection criterion was the availability of a public history of patched vulnerabilities with confirmed CVE identifiers. This ensures that for every sample, not only is the presence of a flaw known, but also its specific CWE taxonomy type, severity level, and exploit description. Specifically, CVE records were extracted from the National Vulnerability Database (NVD) and mapped to corresponding repository commits. Subsequently, "vulnerable" (pre-patch) and "secure" (post-patch) versions of the files were retrieved from the Git history. Annotation was performed at the function and line levels—rather than the file level – to prevent the model from learning to associate vulnerabilities with file names instead of the actual code. Additionally, the OWASP Benchmark and ExploitDB were utilized to cover vulnerabilities that lack a direct CVE but are documented in the OWASP Top 10:2021 [32].

Synthetic data is 37.7% (5,300 records) of the total. For certain CWE classes, the number of real-world vulnerabilities proved insufficient for robust model training. This was particularly evident for rarer types, such as Deserialization of Untrusted Data (CWE-502) or URL Redirection to Untrusted Site (CWE-601). To address this class imbalance, synthetic samples were generated using two distinct methods.

The first method involved syntactic mutations: targeted modifications were automatically introduced into secure functions to transform them into vulnerable ones. For instance, a parameterized SQL query might be replaced with string concatenation (SQL Injection, CWE-89), or a htmlspecialchars() call might be removed from a data output point (XSS, CWE-79). The resulting code mimics realistic scenarios where a developer has committed a common programming error.

The second method utilized HTTP request fuzzing. Requests with mutated parameters, including embedded SQL fragments, XSS payloads, and path traversal sequences were sent to a test application. Responses indicating potential vulnerabilities were captured as positive samples for the DAST component. This approach enriched the dataset with dynamic execution signals alongside static code features.

The complete dataset was randomly partitioned into training, validation, and test sets using a 70% / 15% / 15% ratio. The validation set was used exclusively for hyperparameter tuning, specifically for optimizing the α coefficient in the GNN and LLM blending formula. The test set was reserved solely for the final performance evaluation, which was conducted once after the training of all models was completed.

To evaluate classification performance, several standard metrics derived from the confusion matrix were employed. Precision measures the proportion of correctly identified vulnerabilities among all detected cases, while Recall indicates the model's ability to find all actual vulnerabilities within the dataset. The F1-score serves as a comprehensive indicator by providing a harmonic mean between precision and recall. To assess potential noise, the False Positive Rate (FPR) was used to track the percentage of secure code incorrectly flagged as vulnerable. Finally, the Area Under the ROC Curve (AUC) provides an aggregate measure of the model's ability to distinguish between classes across all possible classification thresholds.

To ensure the statistical significance of the findings, we formulated a null hypothesis (H0) stating that the hybrid model's average performance does not differ from the best-performing standalone LLM. The alternative hypothesis (H1) asserted the superiority of the hybrid architecture. These hypotheses were verified using a one-tailed Student's t-test at a significance level of α is equal to 0.05.

Once all four architectures were trained and their hyperparameters optimized, final performance was assessed using a test set of 2,106 unseen samples. These results are summarized in Table 1.

Table 1
Comparative Performance Analysis of Architectures

Architecture	Precision	Recall	F1-score	ROC-AUC
Random Forest	0,81	0,74	0,77	0,84
GNN	0,86	0,82	0,84	0,89
LLM	0,88	0,85	0,86	0,91
GNN + LLM (гібрид)	0,92	0,89	0,90	0,95

Random Forest, a classic ensemble method typical for traditional SAST scanners, achieved an F1-score of 0,770 and a ROC-AUC of 0,840. While this provides an acceptable baseline, the model struggles with vulnerabilities where context is more critical than formal structure. For instance, it cannot distinguish between a parameterized query and string concatenation when both appear structurally similar in terms of tree-based features.

The GNN improved the F1-score to 0,840, as its data flow graphs allow for tracking how "unsafe" values propagate through various functions toward sensitive operations (sinks). However, GNNs show limited performance in capturing semantics, such as comments, variable names, and general coding style.

The LLM reached an F1-score of 0,860 due to its ability to understand context; for example, the model recognizes when a function handles sensitive payment data and increases its sensitivity accordingly. Nevertheless, LLMs lack an explicit representation of the dependency graph and may overlook vulnerabilities distributed across multiple files that are not visible within a single context window.

The hybrid architecture combined the strengths of both approaches. Its F1-score is 0.90 and its ROC-AUC is 0.95. Statistical tests confirm that the hybrid model significantly outperforms the standalone LLM, proving that these results are not accidental.

Conclusion

This study confirms that the primary challenge facing modern web application security systems is fragmentation rather than a lack of tools. SAST, DAST, and SCA operate in isolation, each using its own format without a shared context, which leads to a surge in false positives and overlooked threats. The proposed ISAVDWA system addresses this problem through three interconnected mechanisms. First, the normalization module consolidates heterogeneous scan results into a unified feature space, eliminating duplicates and contradictions between sources. Second, the hybrid GNN + LLM architecture analyzes code simultaneously at two levels: structural, by tracking data flows through dependency graphs, and semantic, by understanding the logical intent of operations. This synergy resulted in an F1-score of 0.90 and a ROC-AUC of 0.95, outperforming the best standalone approach (LLM), which reached an F1 of 0.86. Third, contextual risk scoring replaces abstract CVSS

scores with real-world priorities; for instance, a vulnerability in a public payment API is correctly identified as more critical than a technically severe defect within a closed test environment.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude AI and ChatGPT in order to: translate and stylistically refine the text, as well as to verify the correctness of grammar, spelling, and technical terminology. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Xygeni, CWE vs CVE: Key Differences Explained, Blog post, 2024. URL: https://xygeni.io/blog/cwe-vs-cve-key-differences-explained/

[2] MITRE Corporation, Common Attack Pattern Enumeration and Classification (CAPEC), Website, 2025. URL: https://capec.mitre.org/

[3] Lan, S., Duan, Z., Lu, S., Tan, B., Chen, S., Liang, Y., & Chen, S. (2024). SLA-ORECS: an SLA-oriented framework for reallocating resources in edge-cloud systems. Journal of Cloud Computing, 2024(1), 18. doi:https://doi.org/10.1186/s13677-023-00561-0

[4] K. Kamran, E. Yeh, & Q. Ma, (2022) Deco: Joint computation scheduling, caching, and communication in data-intensive computing networks. IEEE/ACM Transactions on Networking, 30(3) 1058–1072.

[5] CrowdStrike, Software Composition Analysis (SCA) Explained, Website, 2024. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/cloud-security/software-composition-analysis/.

[6] K. Guo, M. Yang, Y. Zhang, & J. Cao, (2022) Joint computation offloading and bandwidth assignment in cloud-assisted edge computing. IEEE Transactions on Cloud Computing, 10(1) 451–460.

[7] D. H. Nguyen, A. Seo, N. P. Nnamdi, & Y. Son, (2023) False Alarm Reduction Method for Weakness Static Analysis Using BERT Model. Applied Sciences, 13(6) 3502. https://doi.org/10.3390/app13063502

[8] X. Li, L. Wang, Y. Xin, Y. Yang, & Y. Chen, (2020) Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning. Applied Sciences, 10(5) 1692. https://doi.org/10.3390/app10051692

[9] J. Jeong, Y. Son, & Y. Lee, (2019) A Study on the Secure Coding Rules for Developing Secure Smart Contract on Ethereum Environments. International Journal of Advanced Science and Technology, 133 47–58.

[10] S. Zaharia, T. Rebedea, & S. Trausan-Matu, (2022) Machine Learning-Based Security Pattern Recognition Techniques for Code Developers. Applied Sciences, 12(24) 12463. https://doi.org/10.3390/app122412463

[11] MITRE Corporation, CAPEC — Common Attack Pattern Enumeration and Classification, Website, 2025. URL: https://capec.mitre.org/

[12] K. Kuszczyński, & M. Walkowski, (2023) Comparative Analysis of Open-Source Tools for Conducting Static Code Analysis. Sensors, 23(18) 7978. https://doi.org/10.3390/s23187978

[13] Common Weakness Enumeration (CWE). MITRE Corporation. 2025. URL: https://cwe.mitre.org/ (accessed: 03.03.2026).

[14] OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. OWASP Foundation. URL: https://owasp.org/www-project-top-ten/ (accessed: 03.03.2026).

[15] ISO/IEC 27005:2022. *Information security, cybersecurity and privacy protection — Guidance on managing information security risks*. Geneva: ISO/IEC, 2022.

[16] M. Alshomrani, (2024) Survey of Transformer-Based Malicious Software Detection Systems. Electronics, 13(4677). https://doi.org/10.3390/electronics13234677
[17] A. Varga, (2025) A Machine Learning-enhanced web-crawler for vulnerability detection: A binary classification approach. Master's thesis, Blekinge Institute of Technology, 62 p.
[18] T. Zeng et al., (2025) Modern Approaches to Software Vulnerability Detection: A Survey of Machine Learning, Deep Learning, and Large Language Models. Electronics, 14(22). URL: https://www.mdpi.com/2079-9292/14/22/4449
[19] A. Boucena, (2025) Leveraging Large Language Models For Automated Software Vulnerability Detection and Analysis. Master's thesis, University of Guelma, 95 p.
[20] S. Cao, D. Zhu, L. Bo, Y. Wei, & B. Li, (2021) BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection. Information and Software Technology, 136 106576. https://doi.org/10.1016/j.infsof.2021.106576
[21] D. Hin, A. Kan, H. Chen, & M. A. Babar, (2022) LineVD: Statement-level Vulnerability Detection using Graph Neural Networks. Proceedings of the 19th International Conference on Mining Software Repositories (MSR). https://doi.org/10.48550/arXiv.2203.05181
[22] Z. Li, S. Dutta, & M. Naik, (2024) IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities. arXiv:2405.17238. https://arxiv.org/abs/2405.17238
[23] G. Markowsky, O. Savenko, S. Lysenko, & A. Nicheporuk, (2018) The technique for metamorphic viruses' detection based on its obfuscation features analysis. CEUR-WS, 2104 680–687.
[24] O. Savenko, S. Lysenko, A. Nicheporuk, & B. Savenko, (2017) Approach for the Unknown Metamorphic Virus Detection. Proceedings of the 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems, Bucharest, 71–76.
[25] O. Savenko, S. Lysenko, A. Nicheporuk, & B. Savenko, (2017) Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search. CEUR-WS, 1844 555–569.
[26] I. Ramskyi, A. Drozd, O. Lyhun, & O. Ponochovna, (2025) System for cybersecurity evaluation of corporate networks. Computer Systems and Information Technologies, (2) 123–131. https://doi.org/10.31891/csit-2025-2-14
[27] A. Kashtalian, S. Lysenko, A. Sachenko, B. Savenko, O. Savenko, & A. Nicheporuk, (2025) Evaluation criteria of centralization options in the architecture of multicomputer systems with traps and baits. Radioelectronic and Computer Systems, (1) 264–297. https://doi.org/10.32620/reks.2025.1.18
[28] M. Maidan, & A. Melnyk, (2023) Organization of FPGA-based Devices in Distributed Systems. International Journal of Computing, 22(3) 352–359. https://doi.org/10.47839/ijc.22.3.3231
[29] T. Maksymyuk, F. Meloni, M. Torres Diaz, D. Romano, L. Belucci, & N. Chukhray, (2026) Decentralized Blockchain Framework for the Provenance of Cultural Heritage. International Journal of Computing, 24(4) 780–789. https://doi.org/10.47839/ijc.24.4.4345
[30] A. Bomba, M. Nazaruk, N. Kunanets, & V. Pasichnyk, (2017) Constructing the diffusion-like model of bicomponent knowledge potential distribution. International Journal of Computing, 16(2) 74–81. https://doi.org/10.47839/ijc.16.2.883.
[31] O. Savenko, S. Lysenko, A. Kryschuk, Multi-agent based approach of botnet detection in computer systems. In: A. Kwiecień, P. Gaj, P. Stera (eds.), Communications in Computer and Information Science, vol. 291, Springer, Heidelberg, 2012, pp. 171–180. https://doi.org/10.1007/978-3-642-31217-5_19.
[32] H. Booth, D. Rike, & G. Witte, (2023) The National Vulnerability Database (NVD): Overview. National Institute of Standards and Technology (NIST).