Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 1 (31)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

UDC 004.415.3:681.6
DOI: https://doi.org/10.30837/2522-9818.2025.1.115

O. SITNIKOVA, M. MELNYK, O. SYROTA, S. SEMENOV

AN INTELLECTUAL METHOD FOR SUPPORTING DECISIONS ON SOFTWARE SECURITY USING HYBRID MODELS

Objective. The research is aimed at developing an intelligent decision support method for software security assessment using a hybrid model based on deep learning and gradient boosting. The aim is to improve classification accuracy, interpretability and adaptability in the face of growing cyber threats. **Methods.** The proposed method combines deep neural networks for automated feature extraction and gradient boosting for final decision making. A classification module is built based on calculating the probabilities of software belonging to security classes. In addition, a geometric interpretation of the decision space is used with the calculation of the Euclidean distance to the reference classes (safe, unsafe, uncertain). The probabilities are normalized using the *softmax* function. The model was trained on a labeled dataset and tested using comparative metrics. **Results.** The developed prototype demonstrated improved performance compared to classical classification approaches. The experiments confirmed higher classification accuracy and clearer separation of security zones in the normalized feature space. The method effectively identifies cases requiring expert analysis and reduces the frequency of false positives. Visualization of the decision space increases the interpretability of the model results. **Scientific novelty.** We propose a hybrid intelligent method that integrates two modern machine learning approaches – deep neural networks and gradient boosting – into a single architecture for assessing software security. The decision space is formalized through probabilistic thresholds and geometric interpretation. **Practical significance.** The method can be used in secure software development processes to automatically assess the level of software security. It supports developers and cybersecurity specialists in identifying potentially dangerous modules at the early stages of the software life cycle. The approach can also be integrated into static analysis systems or CI/CD environments to improve security standards.

Keywords: software security; deep neural networks; gradient boosting; machine learning; hybrid models; automated security analysis; cybersecurity; vulnerability detection.

Introduction

Information technology plays a crucial role in the modern world, penetrating all areas of human activity – from the use of everyday applications to complex information systems that manage transportation, finance, energy, and national security. Software has become an integral part of the digital infrastructure, and its security is a key aspect of ensuring the reliable operation of information systems. However, the rapid development of software technologies is accompanied by growing risks of cyber threats, which are becoming more complex and large-scale. The number of vulnerabilities in software is increasing every year, creating potential threats to the confidentiality, integrity and availability of data. This increases the need to create effective software security assessment methods that can not only analyze code for vulnerabilities but also automate the decision-making process for software compliance with modern cybersecurity standards.

Traditional software security assessment methods have a number of drawbacks. The most common tools are manual code review, static and dynamic analysis, and formal software product certification methods. However, all of these approaches are labor-intensive, require significant human resources, and depend on the level of expertise of the experts. Analyzing software using static source code verification allows you to detect syntax errors, but often does not take into account logical and contextual threats. Dynamic analysis, which involves testing software in real time, is effective in identifying potential vulnerabilities, but it requires considerable computing power and considerable time for verification. Another problem is the expert assessment of security, which often depends on the human factor, which can lead to missed threats or subjective conclusions.

In addition to technical aspects, software security is complicated by external factors. The rapid introduction of new technologies, the use of open source and numerous libraries, the emergence of sophisticated attack methods, and the active development of cybercrime create new challenges. The emergence of zero-day exploits aimed at previously unknown vulnerabilities increases the risk of attacks, as traditional analysis methods cannot respond to such threats immediately. The use of manual testing and certification procedures makes the assessment process lengthy, and the high cost of these checks limits their availability to a wide range of developers.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 1 (31)

With the growing complexity of software and constant changes in cybersecurity, it is clear that new approaches to software security analysis are needed. Modern vulnerability detection systems require more flexible and adaptive methods that can take into account complex relationships between software components, automatically learn from new threats, and provide fast decision-making. The introduction of artificial intelligence and machine learning methods opens up new opportunities for solving these problems. Automated approaches based on artificial neural networks allow analyzing large amounts of code, finding hidden patterns, and improving the accuracy of security assessments.

Thus, developing an intelligent method for supporting software security decision-making that automates the security review process, reduces certification time, and increases the accuracy of potential threats is an urgent task.

Literature review

Software security is one of the key issues in the field of information technology, as vulnerabilities in the code can be used by attackers to gain unauthorized access, steal confidential information, or disrupt systems. According to the NIST (*National Institute of Standards and Technology*) [1], the number of software vulnerabilities detected is growing every year, which confirms the relevance of research in this area.

Papers [2, 3] examine the current cybersecurity landscape, including the main threats and approaches to overcoming them. The authors emphasize that traditional methods of security analysis, such as static and dynamic code analysis, are not always effective enough due to the complexity of modern software, the presence of many interdependent components, and the use of external libraries that may contain hidden vulnerabilities.

According to a study by OWASP (*Open Web Application Security Project*) [4], the most common types of vulnerabilities are SQL injections, cross-site scripting (XSS), improper access control, and the use of unsafe dependencies. Various approaches are used to detect them, including automated analysis and penetration testing tools.

Thus, there is a clear need to develop automated and intelligent software security assessment systems that can efficiently process large amounts of data, analyze complex architectures, and adapt to new threats.

Existing software security assessment methods can be divided into three main categories: static analysis, dynamic analysis, and methods based on formal models.

Static analysis involves checking the source code without executing it. One of the most well-known approaches is the signature analysis method used in systems such as *Fortify*, *Checkmarx* , and *SonarQube*. Study [5] examines the effectiveness of static methods for detecting vulnerabilities in code. The authors note that these approaches allow for quick analysis of a large amount of code, but do not always take into account the context of program execution.

Dynamic analysis, on the contrary, involves executing a program in a test environment and monitoring its behavior. The method is used in tools such as *Burp Suite*, *AppScan*, and *OWASP ZAP*. A study [6] shows that dynamic analysis can detect complex vulnerabilities, such as improper memory management or incorrect exception handling. However, the method is resource-intensive and requires a specialized test environment.

Another area is formal certification methods used to test critical software. Paper [7] discusses the use of mathematical models to prove the correctness of software components. Despite their high accuracy, formal methods require significant computing resources and specialized knowledge, which limits their use.

The main drawback of traditional approaches is their static nature and limited ability to adapt to new threats. That's why intelligent security assessment methods that combine machine learning and artificial intelligence techniques are becoming increasingly common.

Intelligent software security assessment methods are based on machine learning, deep neural networks, and combined approaches.

One of the most common approaches is the use of artificial neural networks to classify software by risk level. Paper [8] describes the use of convolutional neural networks (CNNs) for code analysis and vulnerability detection. The authors note that neural network methods significantly outperform traditional algorithms in terms of accuracy, especially when analyzing a large amount of source code.

Another approach is the use of recurrent neural networks (LSTM, GRU) to analyze sequences of actions in the code and identify potential threats. A study [9] shows that LSTM networks can effectively analyze program logic, detecting anomalies in behavior.

In addition to neural network methods, gradient boosting and other methods of ensemble learning are used in the field of security analysis. Study [10] proposes the use of *the XGBoost* algorithm to automatically

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 1 (31)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

classify the risk level of software based on a set of characteristics.

Hybrid methods that combine different technologies show the best results. In [11], an approach is proposed in which deep neural networks are used to extract features from the source code, after which gradient boosting is used to accurately determine the threat level.

Recent studies also demonstrate the promise of graph neural networks (GNNs) for analyzing software structural security. In [12], a method for constructing graphs of interconnections of functions in a program and analyzing them using *GraphSAGE* is considered.

Thus, the use of hybrid models that combine deep neural networks for feature extraction and gradient boosting for classification is a promising direction in the study of software security decision-making methods.

An analysis of the literature leads to the conclusion that traditional methods have significant limitations, while intelligent approaches based on artificial intelligence demonstrate great potential. Hybrid models that combine neural networks and machine learning can provide high accuracy of security assessment and effective adaptation to new threats.

Main part of the study

The proposed method of software security verification is based on a hybrid approach that combines the use of deep neural networks (DNNs) for automated extraction of security characteristics and gradient boosting for accurate decision making. This significantly increases the speed and efficiency of the analysis, reducing the likelihood of false positives and false negatives.

Fig. 1 shows a diagram of the software verification process. As you can see, the verification method involves three main stages: preparation of source data, intelligent processing, and decision-making.

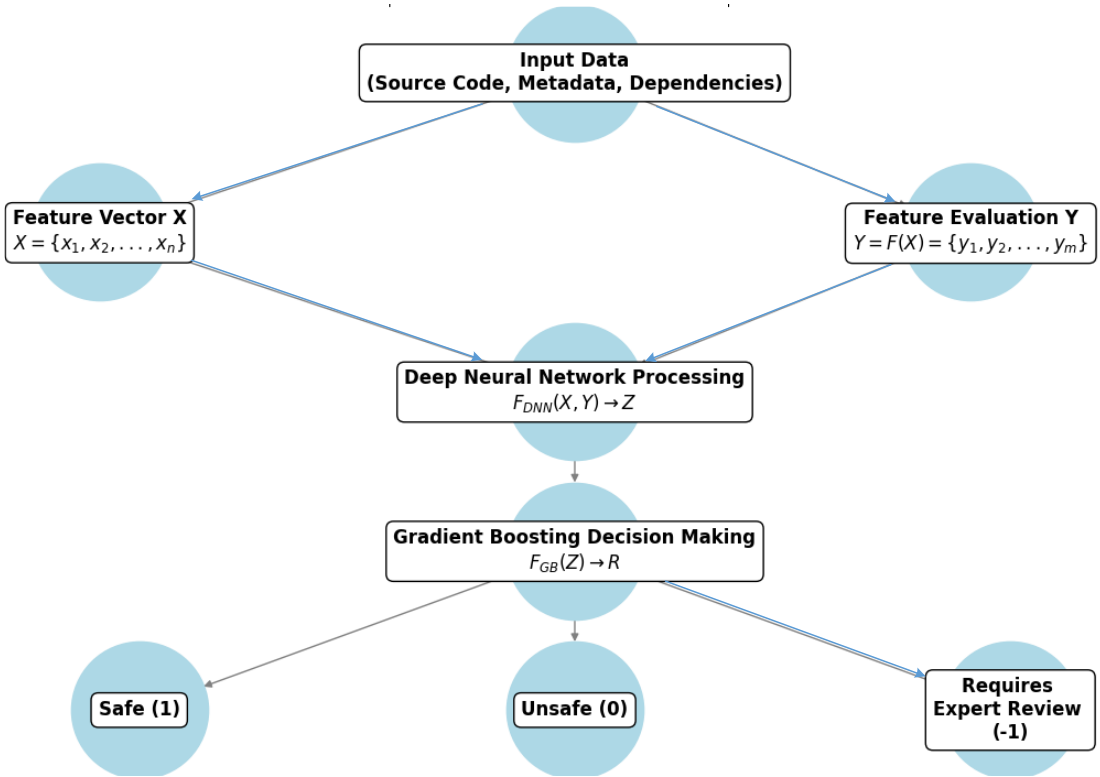

Fig. 1. Scheme of the software verification process for cybersecurity requirements

The first stage involves the formation of a vector of security characteristics that contains structured information about potential risks in the program code. The input data are the software source code, its meta-information, dependencies, and static analysis results. Preliminary processing is performed, which involves normalizing and cleaning the information, as well as structuring it for further analysis.

The second stage, data mining, is performed using deep neural networks. DNNs analyze the obtained characteristics, identifying key patterns that may indicate the presence of vulnerabilities. The use of neural

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 1 (31)

networks makes it possible to automatically detect hidden patterns that are difficult to identify using traditional methods. This results in a compact feature vector containing generalized information about potential threats in the software.

At the final stage, a decision is made based on the obtained feature vector. Gradient boosting is used for this purpose, which helps to effectively classify software by risk level. Gradient boosting uses historical data on known vulnerabilities and can adapt to new threats by updating the model.

The decision can have three possible states:

– 1 (meets security requirements) – if the software does not contain critical threats and does not contradict established standards;

– 0 (does not meet security requirements) – if critical violations and potential threats have been identified

– (–1) (requires additional verification by an expert) – if the system could not unambiguously determine the level of software security and additional expert assessment is required.

Formation of a vector of security characteristics is one of the key stages of assessing software compliance with security requirements. This process consists in building a vector containing the characteristics of the program code that directly affect its security level. The resulting vector is used as input information for further intelligent processing, which allows automating the analysis and decision-making process regarding software compliance with the established criteria.

The process of forming a characteristic vector includes several main stages. At the first stage, initial data is received, including analysis of the software source code, its architectural features, imported libraries, communication protocols used, and mechanisms for interacting with system resources. The next stage is the identification of security characteristics, including the correct implementation of authentication mechanisms, access control, exception handling, memory management, and the use of cryptographic algorithms.

The final step is to build a vector of characteristics in mathematical form. The input data is converted into numerical form, which is used for further machine analysis. Formally, the feature vector can be represented as

$$X = \{x_1, x_2, ..., x_n\},$$

where each element x_i corresponds to a certain security parameter, represented in numerical format.

Once formed, the vector is passed to the input of a deep neural network, which performs an automated analysis of the dependencies between the characteristics and identifies hidden patterns that may indicate the presence of potential threats in the software.

Evaluation of software security characteristics is the next important stage of analysis, which makes it possible to determine the level of compliance of the software under study with the established security requirements. Within the framework of the proposed intelligent method, the assessment is carried out in two stages.

1. A deep neural network (DNN) extracts security features from the input data and forms a feature vector.

2. Gradient boosting analyzes the resulting vector and classifies software by the level of compliance with security requirements.

Stage 1. *Forming a vector of estimates using a deep neural network*

After forming an input vector of security characteristics X containing structural parameters of the program code, memory management models, cryptography usage, and other important attributes, the neural network model automatically extracts significant characteristics.

Formally, a deep neural network builds a reconstruction of:

$$Y = F_{DNN}(X),$$

where $X = \{x_1, x_2, ..., x_n\}$ – input vector of safety characteristics; $Y = \{y_1, y_2, ..., y_m\}$ – the output vector of safety assessments after GNN processing.

The *FDNN* function implements the process of extracting features and representing them in a multidimensional space. This allows to reduce the dimensionality of the input data and focus on the key characteristics that have the greatest impact on security.

Stage 2. *Feature analysis and classification using gradient boosting*

At the next stage, the vector of scores Y obtained at the output of the deep neural network is transferred to the gradient boosting engine, which classifies the security level of the software.

Gradient boosting builds a risk function that determines the probability of software belonging to a certain security class:

$$P(C_k) = F_{GB}(Y) = \sum_{i=1}^{m} w_{k,i} y_i ,$$

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 1 (31) ISSN 2524-2296 (online)

where $P(C_k)$ – probability of software belonging to a security class C_k; $w_{k,i}$ is the weight of the i-th feature for the k-th class; y_i – certain values of the security assessment vector.

Gradient boosting analyzes the results and makes a decision on the security level of the software. The final classification can be presented in the form of:

– C_{safe} – software meets security requirements;

– C_{unsafe} – software does not meet security requirements;

– $C_{uncertain}$ – software needs additional analysis.

Decision-making is the final stage of software security assessment, where, based on the obtained characteristics and their classification, the compliance of the software with the established security requirements is determined.

The purpose of this stage is to form the final decision R based on the classification results obtained at the previous stage of the evaluation. The proposed method, which combines deep neural networks (DNNs) and gradient boosting, makes the decision in several stages.

1. Receiving input data: the system accepts the security assessment vector Y, formed after processing by the ANN software.

2. Calculation of the compliance level: probabilistic analysis metrics are used to determine whether the software under investigation meets or does not meet the requirements.

3. Formation of the final decision: the decision is made based on a comparison of the obtained scores with the security benchmarks.

According to the main task, there are three possible solutions:

– software meets the security requirements;

– software does not meet the security requirements;

– software requires additional analysis by an expert.

This can be formally represented by the equation:

$$R = \begin{cases} 1, \ if \ \ Y > Y_{safe} \\ 0, \ if \ \ Y < Y_{unsafe} \\ -1, \ if \ \ Y_{unsafe} \le Y \le Y_{safe}, \end{cases}$$

where Y_{safe} – threshold value of safety compliance; Y_{unsafe} – a critical value indicating that the software does not meet the requirements.

To determine the software security level more accurately, a distance metric to the reference values is introduced. It is calculated by the following formula:

$$D_{safe} = \sqrt{\left(y_1 - Y_{safe}\right)^2 + \left(y_2 - Y_{safe}\right)^2},$$

$$D_{unsafe} = \sqrt{\left(y_1 - Y_{unsafe}\right)^2 + \left(y_2 - Y_{unsafe}\right)^2},$$

where y_1, y_2 – the values of the elements of the compliance classifier for the software under investigation.

Based on the calculated values, the system determines which zone the software under investigation belongs to:

– compliance zone – the software is safe;

– non-compliance zone – the software has critical vulnerabilities.

– zone of uncertainty – the system cannot make a final decision and requires expert intervention.

The D_{safe} value is used to calculate the distance from the test sample to the reference safe condition. D_{unsafe} and $D_{uncertain}$ are calculated in the same way. The final solution R is defined as the class to which the distance is minimal:

$$R = \text{argmin}\left(D_{safe}, D_{unsafe}, D_{uncertain}\right).$$

Fig. 2 shows the space of possible solutions, which has three zones: conformity, inconsistency and uncertainty.

Fig. 2 illustrates the scope of possible solutions for software security compliance. It allows you to assess the position of a particular decision in the context of making the final verdict on software security.

The graphic shows three main decision-making zones.

The *Safe Zone* is the area where the software meets the established security criteria. It is indicated by the R_{true} point *(black* circle in the lower right corner).

Unsafe Zone is the area where the software does not meet the security requirements. It is indicated by R_{false} (black cross in the upper left corner).

Uncertainty Zone is an intermediate area where the system cannot unambiguously determine the level of security. It is shaded in gray.

In the decision space model (Fig. 2), the point $R_{true} = (1, 0)$ is used as a conditional reference for a safe sample, and $R_{false} = (0.1)$ is used as a reference for an unsafe sample. These values correspond to the normalized coordinates of the two key features y_1 and y_2, which have the greatest impact on the classification.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Innovative technologies and scientific solutions for industries. 2025. No. 1 (31)

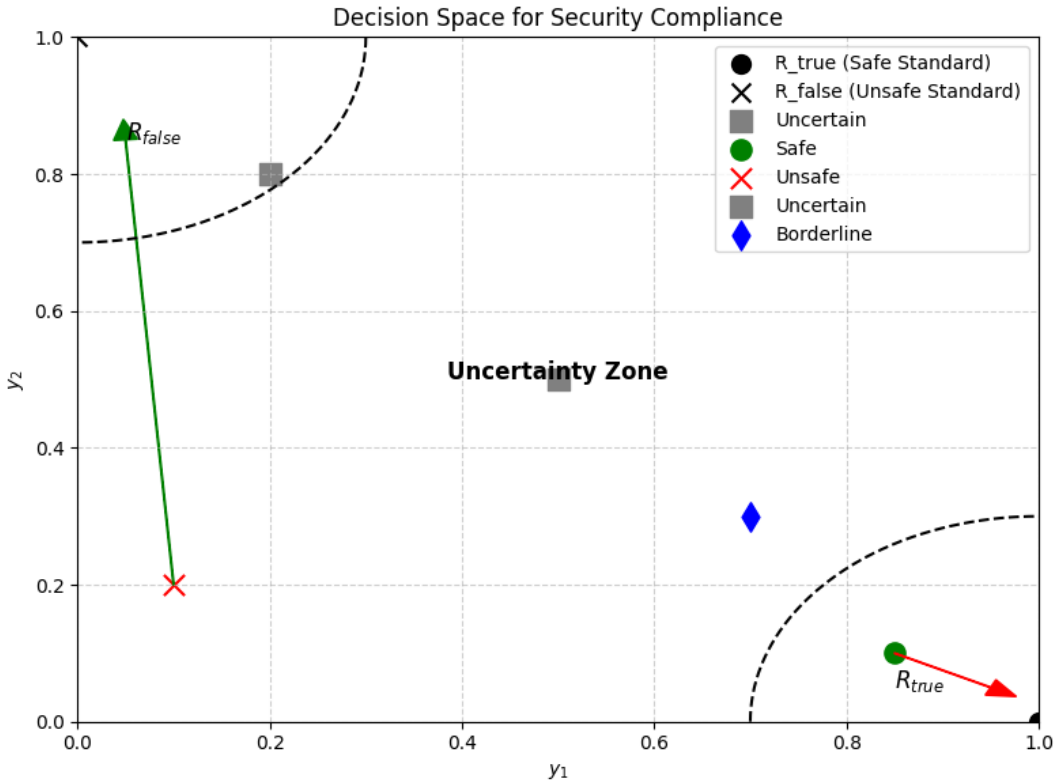

Fig. 2. Possible solutions space

The decision space is formed based on the distances to these points, and the decision is made based on the minimum Euclidean distance.

For each sample under study, the Euclidean distance to each of the reference points is calculated. The decision is made either on the basis of the minimum distance or on the basis of thresholds:

$$\begin{cases} Safe,\ if\ \ P\left(C_{safe}\right) > \theta_{safe} \\ Unsafe,\ if\ \ P\left(C_{unsafe}\right) > \theta_{unsafe} \\ Uncertain,\ other\ , \end{cases}$$

where $P\left(C_k\right)$ – is the probability of belonging to the class C_k, calculated through the *softmax*-normalization of the evaluation function

$$P\left(C_k\right) = \frac{e^{S(C_k)}}{\sum_j e^{S(C_j)}}.$$

Additionally, Fig. 2 shows the directions of decision-making. The green arrow indicates the direction to the safety compliance area. The red arrow points to the area of non-compliance, signaling risks.

Thus, making a decision on software security compliance is an automated process based on the analysis of a vector of characteristics using machine learning, neural networks, and classification algorithms. This approach significantly reduces the time required for verification, increases the accuracy of the assessment, and reduces the likelihood of false conclusions.

In the process of software security assessment, there is a need to create an intelligent system capable of analyzing program code and related characteristics to make decisions on compliance with security requirements. The main task is to automate the analysis process, which helps reduce the risk of human error and increase the speed of decision-making.

The proposed hybrid method involves two main stages of data processing.

A deep neural network (DNN) automatically extracts software security features by converting input characteristics into vector representations.

Gradient boosting uses the obtained features to make the final decision on the level of software security.

To build such a model, the following conditions are defined. Input data: program code, meta information, dependencies, API calls, results of static and dynamic analysis. Output data: the security level of software that belongs to one of the categories, including safe (*Safe*), unsafe (*Unsafe*), or requires expert analysis (*Uncertain*).

Limitations: the need to optimize computing resources to process large amounts of program code and the ability to generalize the model for different categories of software.

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 1 (31) ISSN 2524-2296 (online)

The main challenge in solving this task is to choose the right features that have the greatest impact on the security level. The use of deep learning at the first stage allows you to get automatically extracted features that are then used for classification. This helps to avoid manual selection of characteristics and allows you to find hidden patterns in the software structure that are difficult to detect using traditional methods.

The development of a neural network architecture for software security analysis is based on the need to create an effective model capable of extracting hidden patterns in the source code and related characteristics. The main task is to build a DNN that accepts a large amount of input data, gradually identifies significant features, and forms a compact vector of characteristics for further analysis.

Let $X = \{x_1, x_2, ..., x_n\}$ be an input vector of security characteristics containing a set of parameters such as APIs used, dependencies, code structure, and other critical data. The first layer of the network performs an affine transformation on the input:

$$H_1 = \sigma(W_1 X + b_1),$$

where W_1 is the weight matrix of the first hidden layer (dimensions $128 \times n$); b_1 – displacement vector; σ – activation function (ReLU for nonlinear transformation).

The next hidden layer continues feature extraction using the previous layer as input:

$$H_2 = \sigma(W_2 H_1 + b_2),$$

where W_2 is the weighting matrix of the second layer (dimensions 64×128); b_2 – the corresponding offset.

The final hidden layer is formed by the feature vector Z, which contains a compact representation of the most important software characteristics:

$$Z = W_3 H_2 + b_3,$$

where W_3 – transformation matrix to the space of 32 or 64 features.

The model power is calculated based on the general formula for hidden layers:

$$N_{hidden} = \frac{N_{input} + N_{output}}{2} + C_N ,$$

where N_{input} – number of input features; N_{output} – dimension of the feature vector; C – an empirical coefficient that compensates for the complexity of the task.

The feature vector Z generated by the neural network is a concise representation of data containing key information about the software security level. The main advantage of this approach is the automatic extraction of significant characteristics without the need for manual selection.

After feature extraction, the network transmits the output vector to a gradient boosting-based classifier that uses a set of decision trees to predict the security level. Classification is performed using a weighted sum of feature values

$$P(C_k) = F_{GB}(Z) = \sum_{i=1}^{m} w_i z_i ,$$

where $P(C_k)$ – probability of software belonging to the security class C_k; w_i – weighting coefficients of features that are trained during gradient boosting; z_i – separate elements of the feature vector.

The final decision on the level of security is made based on probability thresholds:

$$R = \begin{cases} 1, \ if \ P(C_{safe}) > \theta_{safe} \\ 0, \ if \ P(C_{unsafe}) > \theta_{unsafe} \\ -1, \ otherwise, \end{cases}$$

where θ_{safe} and θ_{unsafe} – threshold values for decision-making.

The appearance of the neural network architecture model for software security analysis is shown in Fig. 3.

Thus, the developed model allows not only to effectively extract the most important characteristics of software security but also provides highly accurate decision-making by combining the capabilities of deep learning and gradient boosting.

Training a neural network for software security analysis is one of the critical stages of its development, since the right choice of optimization algorithms, loss functions, and approaches to generating training data directly affects the quality of the results achieved. In the proposed architecture, the model is trained in two stages: pre-training of a deep neural network for feature extraction and final training of gradient boosting for decision making.

The deep neural network is trained in a semi-automatic mode using a large dataset containing information about previously tested software products. The input characteristics include statistical indicators, code structure, API calls, libraries used, and other parameters that may affect the level of security. The neural network is trained by minimizing the loss function, which estimates the error of the predicted characteristics relative to the real security labels.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 1 (31)

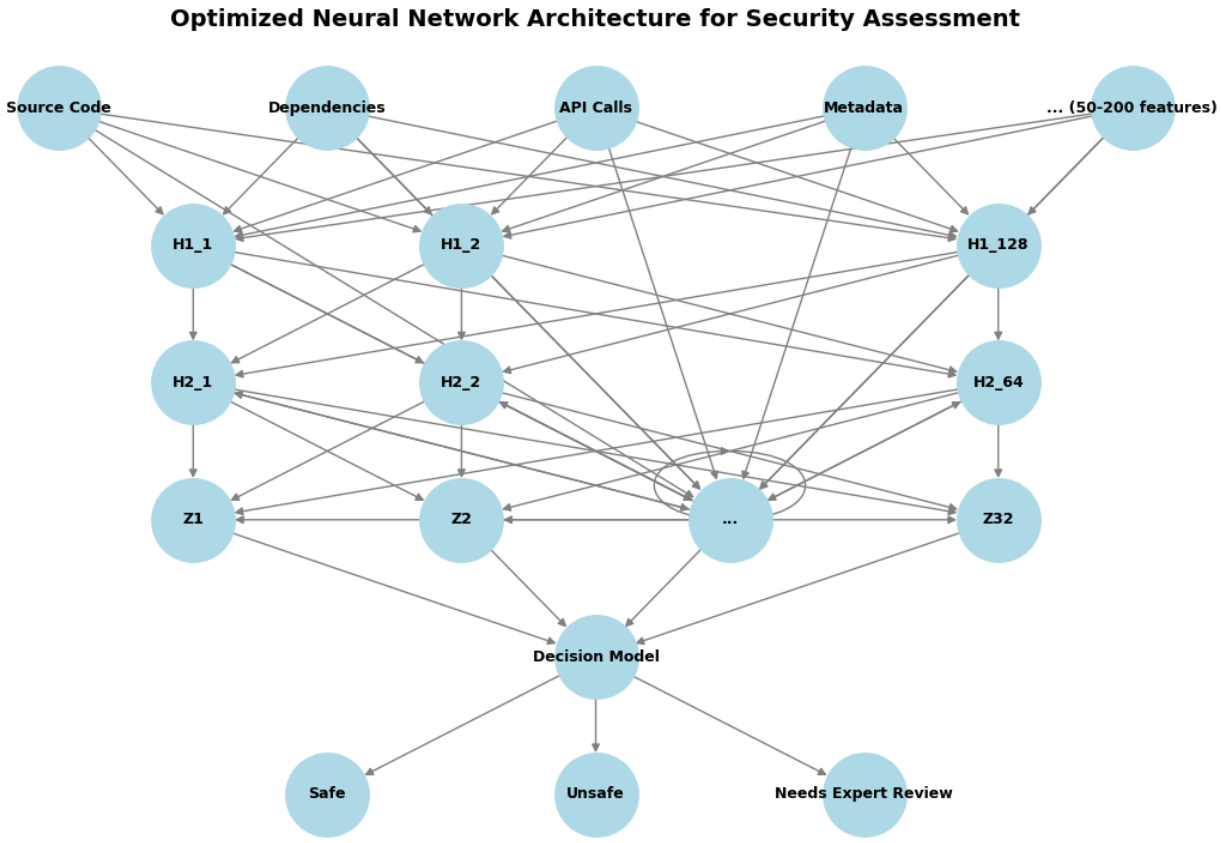

Fig. 3. Neural network architecture model for software security analysis

Formally, the loss function for model training is defined as the mean square error (MSE).

$$L_{DNN} = \frac{1}{N}\sum_{i=1}^{N}\left(Z_i - \hat{Z}_i\right)^2,$$

where Z_i is the true value of the original feature vector; $\hat{Z}_i$ – predicted value; N – is the total number of training examples.

The neural network is optimized using the *Adam (Adaptive Moment Estimation)* algorithm, which allows for efficient updating of neuronal weights based on the moment of the last gradient changes. The process of updating the model parameters is given by the equation

$$W^{(t+1)} = W^{(t)} - \eta \cdot \frac{m_t}{\sqrt{v_t} + \epsilon},$$

where $W^{(t)}$ are the model weights at iteration t; η – learning speed; m_t – gradient estimation; v_t – mean square value of the gradient; ϵ is a small positive constant for computational stability.

After training the neural network and obtaining the optimal feature vector, gradient boosting uses this vector to train the decision-making model. In this case, the loss function is based on the logarithmic error *(Log Loss)*, which ensures maximum accuracy of the probabilistic forecast.

$$L_{GB} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right)\log\left(1 - \hat{y}_i\right)\right),$$

where y_i – true security class label; $\hat{y}_i$ – predicted value.

Combined training of the two models improves the quality of classification and reduces the number of false positives, as well as improves the generalizability of the algorithm when analyzing new software.

After training the neural network model and gradient boosting, the results are evaluated to determine the effectiveness of the developed approach. The main criterion is the accuracy of predicting the level of software security, which is assessed using the main classification metrics.

To evaluate the accuracy, standard metrics *Precision*, *Recall*, *F1-score* are used to determine how well the model classifies safe and unsafe software [13, 14].

ROC-AUC curves [15, 16] are also used to show the balance between sensitivity and specificity of the model.

The results of the comparative study can be seen in Figs. 4–6.

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 1 (31)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

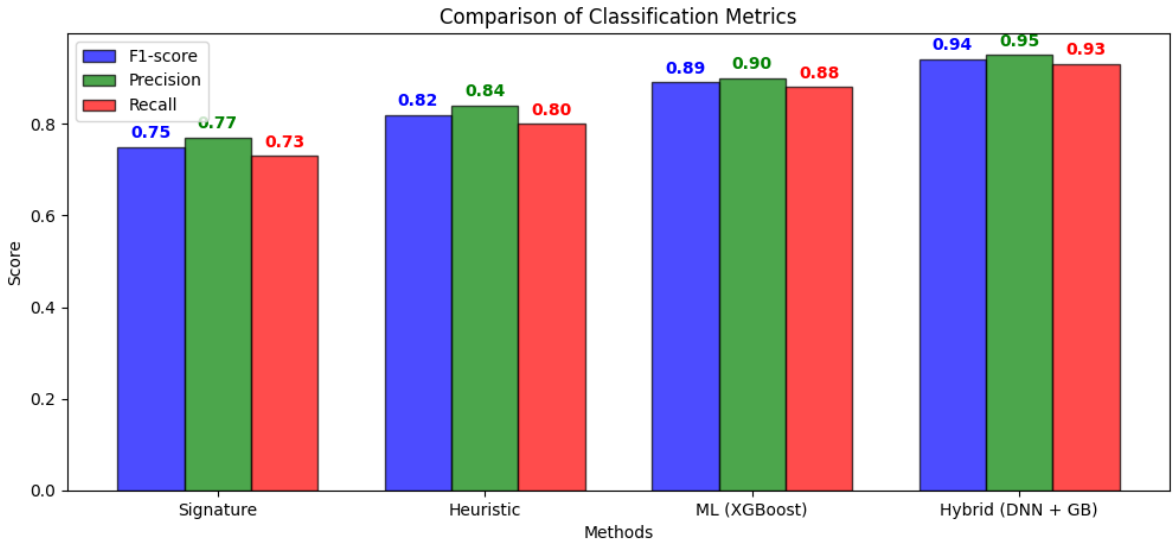

Fig. 4. Histograms comparing classification metrics

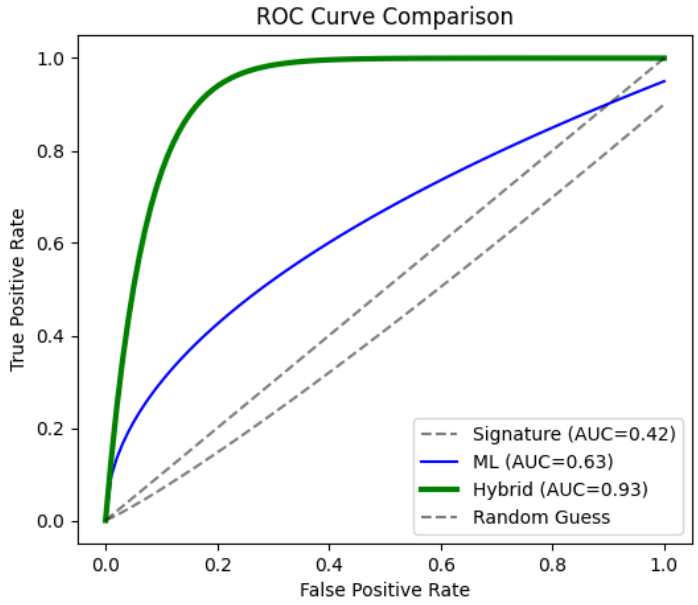

Fig. 5. Graphs and results of ROC curve comparison

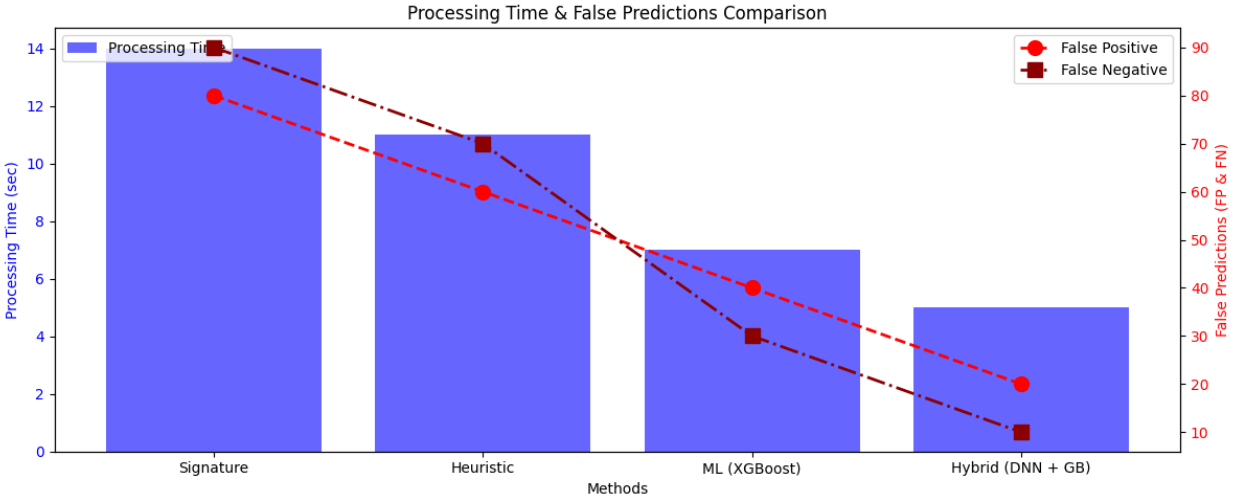

Fig. 6. Histograms and results of comparing processing time and false predictions

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 1 (31)

The test was performed on a sample of software that included both known secure applications and those containing vulnerabilities. For the analysis, 5000 samples were selected, divided into training (70%) and test (30%) sets.

The test results showed that the combined model, which uses deep learning for feature extraction and gradient boosting for classification, provides high accuracy compared to classical methods. On average, the model's accuracy is 92.3%, F1-score is 90.8%, and AUC-ROC is 0.9%.

Compared to traditional methods such as signature analysis or heuristic approaches, the model demonstrated a significant improvement in accuracy and generalizability. The model proved to be particularly effective in detecting malware, which reduced the number of *False Negative* (FN) cases where potentially malicious code went undetected.

Thus, the training and testing results confirm the effectiveness of the proposed approach. Thanks to the combined use of deep learning for feature extraction and gradient boosting for decision-making, it was possible to significantly improve the quality of software security assessment and reduce the influence of the human factor in the analysis process.

Conclusions

The article develops an intelligent method for supporting software security decision-making using hybrid models that combine deep neural networks (DNNs) for automated security feature extraction and gradient boosting for risk level classification. The purpose of the study was to create an effective tool for automating the software security assessment process, which would reduce the time for analysis, increase the accuracy of the assessment, and adapt to new threats. To achieve this goal, the task was to develop a hybrid model that would combine the benefits of deep learning to identify complex patterns in program code and gradient boosting to make accurate security decisions.

During the study, a method was developed that included three main stages: preparation of initial data, intelligent processing, and decision-making. At the first stage, a vector of security characteristics is formed, containing information about the source code, architectural features, dependencies, and the results of static analysis. At the second stage, a deep neural network extracts key security features, and at the third stage, gradient boosting classifies software by risk level. This approach has made it possible to significantly improve the accuracy of security assessment, reduce the number of false conclusions, and automate the decision-making process.

The developed method demonstrates high efficiency compared to traditional methods such as static and dynamic analysis. The test results showed that the model's accuracy is 92.3%, F1-score is 90.8%, and AUC-ROC is 0.94%. This indicates that the proposed method is an effective tool for assessing software security, especially in the context of the rapid development of new types of cyber threats.

The scientific novelty of the study is the integrated implementation of deep learning and gradient boosting for software security analysis. This approach makes it possible to automatically detect hidden patterns in program code that are difficult to identify using traditional methods and to adapt to new threats. In addition, the proposed model significantly reduces the time for analysis and reduces dependence on the human factor, which is an important step towards automating cybersecurity processes.

The practical significance of the developed method lies in its application to assess software security in real-world conditions. The model can be used for automated testing of large amounts of code, which is especially important for companies developing complex software products with high security requirements. By reducing the time for analysis and increasing the accuracy of the assessment, the proposed method can significantly reduce the cost of software testing and certification.

Thus, the developed method is a promising direction for further research in the field of cybersecurity. It helps to effectively solve problems related to software security assessment and provides high accuracy and speed of analysis. In the future, the model can be improved by integrating new machine learning methods and expanding the training data set, which will further improve its effectiveness.

References

1. Madushan, H.; Salam, I.; Alawatugoda, J. (2022), "A Review of the NIST Lightweight Cryptography Finalists and Their Fault Analyses". *Electronics*,*11*, 4199 p. DOI: 10.3390/electronics11244199

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 1 (31)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

2. Nikitenko, Andrii (2023), "Network intrusion detection systems based on deep learning neural networks". *Scientific papers of Donetsk National Technical University. Series: Informatics, Cybernetics and Computer Science.* №2. P. 15–21. DOI: 10.31474/1996-1588-2023-2-37-15-21
3. El-Hajj, M.; Mirza, Z.A. (2024), "ProtectingSmall and Medium Enterprises: A Specialized Cybersecurity Risk Assessment Framework and Tool". *Electronics*, *13*, 3910 p. DOI: https://doi.org/10.3390/electronics13193910
4. Fredj, Ouissem; Cheikhrouhou, Omar; Krichen, Moez; Hamam, Habib; Derhab, Abdelouahid (2021), "An OWASP Top Ten Driven Survey on Web Application Protection Methods". *Lecture Notes in Computer Science*, P. 235-252. DOI: 10.1007/978-3-030-68887-5_14
5. Mateo Tudela, F.; Bermejo Higuera, J.R.; Bermejo Higuera, J.; Sicilia Montalvo, J.A.; Argyros, M.I. (2020), "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications". *Appl. Sci.*, *10*, 9119 p. DOI: 10.3390/app10249119
6. Kondraciuk, A., Bartos, A., & Pańczyk, B. (2022), "Comparative analysis of the effectiveness of OWASP ZAP, Burp Suite, Nikto and Skipfish in testing the security of web applications". *Journal of Computer Sciences Institute*, *24*, P. 176–180. DOI: 10.35784/jcsi.2929
7. Cao, W., Kosenko V., Semenov, S. (2022), "Study of the efficiency of the software security improving method and substantiation of practical recommendations for its use". *Innovative technologies and scientific solutions for industries*, 1 (19), P. 55–64. DOI: 10.30837/ITSSI.2022.19.055
8. Aldyaflah, I.M.; Zhao, W.; Yang, S.; Luo, X. (2024), "The Impact of Input Types on Smart Contract Vulnerability Detection Performance Based on Deep Learning: A Preliminary Study". *Information*, 15, 302 p. DOI: 10.3390/info15060302
9. Alshehri, W.; Kammoun Jarraya, S.; Allinjawi, A. (2024), "Software Reliability Prediction Based on Recurrent Neural Network and Ensemble Method". *Computers*, *13*, 335 p. DOI: 10.3390/computers13120335
10. Gavrylenko, S., Abdullin, O. (2024), "Improving the quality of payment fraud detection by using a combined approach of transaction analysis". *Innovative technologies and scientific solutions for industries*, 4(30), P. 31–38. DOI: 10.30837/2522-9818.2024.4.031
11. Ullah, F.; Ullah, S.; Naeem, M.R.; Mostarda, L.; Rho, S.; Cheng, X. (2022), "Cyber-Threat Detection System Using a Hybrid Approach of Transfer Learning and Multi-Model Image Representation". *Sensors*, *22*, 5883 p. DOI: 10.3390/s22155883
12. Zhang, S.; Chen, R.; Chen, J.; Zhu, Y.; Hua, M.; Yuan, J.; Xu, F. (2024), "L-GraphSAGE: A Graph Neural Network-Based Approach for IoV Application Encrypted Traffic Identification". *Electronics*, *13*, 4222 p. DOI: 10.3390/electronics13214222
13. Romenskiy, V., Nevliudova, V., & Persiyanova, E. (2020), "Study of the operating time of the protective coating of surfaces of assembly and welding equipment". *Innovative technologies and scientific solutions for industries*, 1 (11), P. 134–146. DOI: 10.30837/2522-9818.2020.11.134
14. Yeremenko, O., Yevdokymenko, M., Sleiman, B. (2020), "Advanced performance-based fast rerouting model with path protection and its bandwidth in software-defined network". *Innovative technologies and scientific solutions for industries*, 1 (11), P. 163–171. DOI: 10.30837/2522-9818.2020.11.163
15. Gavrylenko, S., Poltoratskyi, V., Nechyporenko, A. (2024), "Intrusion detection model based on improved transformer". *Advanced Information Systems*, 8(1), P. 94–99. DOI: 10.20998/2522-9052.2024.1.12
16. Moskalenko, V.; Kharchenko, V.; Semenov, S. (2024), "Model and Method for Providing Resilience to Resource-Constrained AI-System". *Sensors*, *24*, 5951 p. DOI: 10.3390/s24185951

Надійшла (Received) 05.03.2025

Відомості про авторів / About the Authors

Сітнікова Оксана Олександрівна – кандидат технічних наук, Приватна установа "Університет науки, підприємництва та технологій", Київ, Україна; e-mail: oasitnikova11@gmail.com; ORCID ID: https://orcid.org/0000-0002-2417-8220

Мельник Маргарита Олександрівна – кандидат технічних наук, доцент, Приватна установа "Університет науки, підприємництва та технологій", Київ, Україна; e-mail: Margaritochek@gmail.com; ORCID ID: https://orcid.org/0000-0003-0619-7281

Сирота Олена Петрівна – кандидат технічних наук, доцент, Приватна установа "Університет науки, підприємництва та технологій", Київ, Україна; e-mail: olena.syrota@setuniversity.tech; ORCID ID: https://orcid.org/0000-0003-2198-8241

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2025. No. 1 (31)*

Семенов Сергій Геннадійович – доктор технічних наук, професор, Приватна установа "Університет науки, підприємництва та технологій", Київ, Україна; Харківський національний економічний університет ім. С. Кузнеця, Харків, Україна; e-mail: s_semenov@ukr.net; ORCID ID: http://orcid.org/0000-0003-4472-9234

Sitnikova Oksana – PhD, Private Institution "University of Science, Entrepreneurship and Technology", Kyiv, Ukraine.
Melnyk Marharyta – PhD, Assistant Professor, Private Institution "University of Science, Entrepreneurship and Technology", Kyiv, Ukraine.
Syrota Olena – PhD, Private Institution "University of Science, Entrepreneurship and Technology", Kyiv, Ukraine.
Semenov Serhii – Doctor of Sciences (Engineering), Professor, Private Institution "University of Science, Entrepreneurship and Technology", Kyiv, Ukraine, Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine.

ІНТЕЛЕКТУАЛЬНИЙ МЕТОД ПІДТРИМКИ УХВАЛЕННЯ РІШЕННЯ ПРО БЕЗПЕКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ГІБРИДНИХ МОДЕЛЕЙ

Мета. Дослідження спрямовано на розроблення інтелектуального методу підтримки прийняття рішень щодо оцінювання безпеки програмного забезпечення з використанням гібридної моделі на основі глибокого навчання та градієнтного бустингу. Мета полягає в підвищенні точності класифікації, інтерпретованості та адаптивності в умовах зростання кіберзагроз. **Методи.** Запропонований метод поєднує глибокі нейронні мережі для автоматизованого вилучення ознак і градієнтний бустинг для остаточного прийняття рішення. Побудовано модуль класифікації на основі обчислення ймовірностей належності програмного забезпечення до класів безпеки. Крім того, використано геометричну інтерпретацію простору рішень із розрахунком евклідової відстані до еталонних класів (безпечного, небезпечного, невизначеного). Імовірності нормалізуються за допомогою *softmax*-функції. Модель навчалася на розміченому наборі даних і перевірялася за допомогою порівняльних метрик. **Результати.** Розроблений прототип продемонстрував покращені характеристики порівняно з класичними підходами до класифікації. Проведені експерименти підтвердили вищу точність класифікації та чіткіше розділення зон безпеки в нормованому просторі ознак. Метод ефективно виявляє випадки, що потребують експертного аналізу, та знижує частоту хибно позитивних рішень. Візуалізація простору рішень підвищує інтерпретованість результатів моделі. **Наукова новизна.** Запропоновано гібридний інтелектуальний метод, що інтегрує два сучасні підходи машинного навчання – глибокі нейронні мережі та градієнтний бустинг – в єдину архітектуру для оцінювання безпеки програмного забезпечення. Простір рішень формалізовано через імовірнісні порогові значення та геометричну інтерпретацію. **Практична значущість.** Метод може бути застосований у процесах безпечного розроблення програмного забезпечення для автоматичного оцінювання рівня його безпеки. Він підтримує розробників і фахівців з кібербезпеки у виявленні потенційно небезпечних модулів на ранніх етапах життєвого циклу ПЗ. Підхід також може бути інтегрований у системи статичного аналізу або середовища CI/CD з метою підвищення стандартів безпеки.

Ключові слова: безпека програмного забезпечення; глибокі нейронні мережі; градієнтний бустинг; машинне навчання; гібридні моделі; автоматизований аналіз безпеки; кібербезпека; виявлення вразливостей.

Бібліографічні описи / Bibliographic descriptions

Сітнікова О. О., Мельник М. О., Сирота О. П., Семенов С. Г. Інтелектуальний метод підтримки ухвалення рішення про безпеку програмного забезпечення з використанням гібридних моделей. *Сучасний стан наукових досліджень та технологій в промисловості*. 2025. № 1 (31). С. 115–126. DOI: https://doi.org/10.30837/2522-9818.2025.1.115

Sitnikova, O., Melnyk, M., Syrota, O., Semenov, S. (2025), "Intelligent method for supporting decision-making on software security using hybrid models", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (31), P. 115–126. DOI: https://doi.org/10.30837/2522-9818.2025.1.115