scientific reports

Check for updates

OPEN

Optimization for threat classification of various data types-based on ML model and LLM

Chaerim Hong & Taeyeon Oh✉

With the development of AI technology, the number of cyber security threats that exploit it is increasing rapidly, and it is urgent to build an effective security threat detection system to respond to these threats. There is active research on AI-based security tools to detect and respond to these security threats. This study explores how heterogeneous data, such as signs of security attacks from security threat news and weaknesses in source code, can be analyzed integrally in an ML model and LLM environment. In this study, we applied scaling and normalization techniques to the Post News data to improve bias, and we used syntax analysis, semantic analysis, and data flow information to perform an integrated analysis of the source code to improve detection performance. It is designed to be applied to both ML models and LLM by systematizing data labeling and data formats. The results showed that the constructed learning model performed well in both text analysis and source code analysis. In the post-news data learning, the ML-based models XGBoost, SVM, and Random Forest all showed f1-scores of 0.96 to 0.97, while the LLM-based models ST5-xxl, XLNet, BERT, CodeBERT, and GraphCodeBERT all showed a score of 0.97. Additionally, in the C/C++ weakness code detection data learning, the LLM series model ST5-xxl achieved 0.9999, XLNet achieved 0.9999, BERT achieved 0.9037, CodeBERT achieved 0.9999, and GraphCodeBERT achieved 0.9999. The ML-based model XGBoost showed an accuracy of 0.9999 with the TF-IDF embedding method, SVM showed 0.9699 with the TF-IDF embedding method, and Random Forest showed 0.9493 with the TF-IDF method. The models demonstrated higher performance with the TF-IDF embedding method than with the Word2Vec embedding. This study proposed an ML and LLM integrated framework that could effectively detect source code vulnerabilities using abstract syntax trees (AST). This framework overcame the limitations of existing static analysis tools and improved detection accuracy by simultaneously considering the structural characteristics and semantic context of the code. In particular, by combining AST-based feature extraction with LLM's natural language understanding capabilities, it improved generalization performance for new types of vulnerabilities and significantly reduced false positives.

Keywords Post news, Data bias, Large language models, Security weakness, Coode weakness, Machine learning

Background

With the advancement of AI technology, the number of cyber security threats that exploit it is increasing rapidly, and it is urgent to build an effective security threat detection system to respond to these threatsn[1,2]. Specifically, there were 604 organisations that had suffered data breaches from 2023 to February 2024, and these breaches had ranged from 2100 to 113,000. The average cost of a data breach had been $4.88 million year-on-year. These results showed that organizations were increasingly demanding security AI and automated responses. Among data breaches caused by security threats, the proportion of personal data breaches had accounted for 46%. Malicious insider attacks had cost a global average of $4.99 million. The highest data breach costs had been in the United States. The industries with the highest data breach costs had been healthcare and finance. There is active research on AI-based security tools to detect and respond to these security threats[3]. There are studies on online abstraction or text summarization systems using deep learning technology, as well as graph-based studies[4,5]. Abstract summarizers use TextRank-based extractive summarizers and bidirectional RNN-based abstract summarizers. Along with research on machine learning in the field of cyber security attack detection, research on the examination of heterogeneous data sets of various types is required[6]. The algorithms used in machine learning are SVM, XGBoost, and Random Forest, and the data sets are NSL-KDD, a data set that has been

Seoul AI School, aSSIST University, Seoul 03767, South Korea. ✉email: tyoh@assist.ac.kr

balanced and removed data duplication but has some data imbalance, and CICIDS-2017, a dataset that includes normal and abnormal traffic and contains class imbalance; UNSW-NB15, a dataset that includes 10 types of attacks and has some class imbalance; and CSE-CIC-IDS2018, a machine learning-based intrusion detection dataset collected in the AWS cloud environment. SVM has high accuracy but requires a long learning time and is difficult to classify multiple classes. Random Forest is an ensemble model that combines multiple decision trees and provides high accuracy but consumes a lot of memory. XGBoost is a high-performance model that uses gradient boosting and has high computational costs and is difficult to interpret. In the study, the imbalance problem was not applied unless the imbalance of the classes between normal and abnormal traffic was severe in the dataset. In order to overcome the limitations of existing machine learning classifiers, Zhou et al. applied the CFS-BA heuristic algorithm when selecting features and combined it with the Random Forest algorithm to detect attacks[7]. They proposed the CFS-BA ensemble method for detecting signs of attacks. However, the research lacks a solution to the problem of data imbalance between attacks and normal traffic. Maniriho et al. apply machine learning algorithms and ensembles to detect signs of attacks and use the NSL-KDD dataset[8]. However, the research lacks approaches to feature selection or dimensionality reduction to improve performance in high-dimensional data problems. Yilaz's research focuses on minimizing the false positive and false negative rates and presents the high accuracy of the optimized model. However, it lacks an approach to unsupervised learning or the application of ensemble models[9].

The importance, method of research includes

This study is significant in the following three aspects. Firstly, it is important as it approaches the prediction of cybersecurity threats from an integrated perspective by comprehensively analyzing security threat information on social media and security vulnerabilities in source code. By presenting a methodology for the integrated analysis of data with heterogeneous characteristics of text and code, it contributes to enhancing the accuracy of security threat detection. Secondly, in terms of data processing, we proposed a method that reflects the characteristics of social media data through scaling and normalization techniques, and improved the accuracy of source code vulnerability detection through Abstract Syntax Tree (AST)-based multidimensional analysis. Thirdly, this study introduced a standardized data processing scheme that can be applied to both the latest language models and traditional machine learning models. The reliability of the research was ensured by utilizing publicly recognized security threat datasets, such as the weakness codes provided by CybAttT and NIST's Juliet. The solution to this problem involves an integrated approach combining machine learning (ML) models and large language models (LLMs). This is necessary due to the differing performance characteristics based on data type and processing environment. ML models are efficient in processing domains with lightweight data, whereas LLMs excel in areas requiring complex contextual understanding. This study analyzes the characteristics of each model and proposes an effective integration strategy. Firstly, for ML models, XGBoost demonstrates excellent classification performance across various data types but presents challenges in result interpretation and sensitivity to outliers. To mitigate these issues, scaling and normalization techniques have been introduced. Support Vector Machines (SVM) are effective for high-dimensional data classification but face limitations when processing large datasets, necessitating performance enhancements through embedding techniques. Random Forest is robust to data noise but has high memory usage, emphasizing the importance of optimized parameter settings. Secondly, regarding LLMs, BERT effectively identifies the contextual meaning of text but does not adequately capture the structural characteristics of source code. To address this, incorporating syntax information through call functions, roles, and Abstract Syntax Tree (AST) encoding is necessary. CodeBERT and GraphCodeBERT specialize in code analysis but encounter limitations in processing data flow complexity and structural information. ST5-xxl and XLNet are strong in natural language processing but show weaknesses in source code analysis, which can be improved by pre-encoding processing for AST columns. Based on the analysis, this study proposes the following integrated approach. First, enhance ML model performance through scaling and normalization during the data preprocessing stage. Second, introduce an encoding technique that combines AST, function, and function role for structural information processing. Third, perform optimized parameter settings when selecting a model by reviewing data characteristics and processing environment. In conclusion, an integrated approach that considers the strengths and weaknesses of both ML models and LLMs is essential for effective analysis of social media and source code data. This research aims to contribute to overall performance improvement by leveraging the complementary strengths and addressing the limitations of the models presented in Table 1.

The distinctive contribution in research

This study introduced a groundbreaking approach to cybersecurity by being the first to simultaneously analyze heterogeneous security threat data from both social media and source code vulnerabilities. While previous research typically addressed these domains separately, our work pioneered their integration through a novel hybrid ML-LLM framework. We developed a distinctive dual-analysis methodology that uniquely bridged the gap between textual threat intelligence and code vulnerability detection. Unlike existing approaches that treated these as separate domains, our technique leveraged their complementary nature to achieve superior threat detection capabilities. We innovatively addressed Post News data bias through a custom weight optimization technique, achieving a breakthrough in handling underrepresented security threat classes. Our ratio-based weighting approach with XGBoost and TF-IDF embedding delivered unprecedented performance metrics, surpassing previous methods by significant margins. For code vulnerability analysis, we created a first-of-its-kind method that uniquely combined function call-centric AST extraction with advanced code graph analysis. This novel approach effectively solved the long-standing scalability and semantic relationship challenges that limited previous methods. Our implementation with GraphCodeBERT and ST5-xxl models achieved near-perfect accuracy rates (> 0.9999), establishing a new state-of-the-art benchmark. We pioneered a tri-dimensional analytical approach integrating syntax, semantics, and information flow—a combination previously unexplored

Model	Strengths	Limitations	References	Model solution
XGBoost	Excellent performance in classifying various data types among ML models Parallel processing possible	Difficult to interpret results due to model complexity Performance degradation due to sensitivity to outliers	7,9,11,12	Solves the problem of data bias in the process of building a data set through scaling and normalization techniques Designed to improve performance in XGBoost, SVM, and Random Forest models using embedding techniques
SVM	Provides effective classification in high-dimensional data High classification performance with optimal setting of class boundaries Partial parallel processing available	When using a date set, there is a lot of learning time and memory usage Limited to multi-class classification The amount of computation increases depending on the size and complexity of the data, making it optimal for parallel processing	7,9,11,12	
Random Forest	Resilient to data noise Provides stable prediction performance Parallel processing possible	High memory usage due to the use of many trees High time consumption when the data set is large May be sensitive to outliers	7,9,11,12	
BERT	Identify the meaning of words accurately by considering the context of the text in the sentence	Failure to reflect structural data characteristics such as source code	13,19,18,23–25	Overcomes the limitations of BERT models by learning through call function, call func role, and AST encoding processing, reflecting syntax information, semantic information, and information flow
Code BERT	Parallel learning of code and natural language is possible	Insufficient reflection of data flow or structural information in the code	13,19,18,23–25	When applying the CodeBERT model, the data flow and structural information of the code are reflected in the learning to solve the problem
Graph Code BERT	Use structural information and understand the meaning of the code	Increased model complexity in structural information processing	13,19,18,23–25	GraphCodeBERT solved the problem of complexity and long learning time by simplifying call function, call function role, and AST information before learning and then learning by constructing a dataset
ST5-xxl	Natural language processing is possible by integrating text conversion issues	Limited to source code analysis Requires a lot of resources for learning and reasoning	25,27	The ST5-xxl model, which is strong in natural language processing, solves the problem of learning complexity by encoding call functions, call function roles, and AST columns during source code analysis and structuring them in a dictionary
XLNet	Better than BERT in text classification	Lack of structural analysis of source code Complex learning process and high computational cost due to permutation combinations	25,27	To solve the problem of the complex learning process and high computational cost of the XLNet model, encoding processing of all func, call func role, and AST columns is performed when analyzing source code to solve the complexity problem in learning

Table 1. Characteristics of models: strengths, limitations, reference, and model solution.

in security threat detection. This unique multi-faceted analysis enabled our models to capture subtle vulnerability patterns missed by conventional single-dimension approaches.

Our research team developed the first comprehensive framework for the integrated analysis of diverse security data types. Unlike previous siloed approaches, we created techniques that simultaneously analyzed social media signals and code vulnerabilities, establishing a new paradigm in security threat detection. We introduced a novel class-balancing methodology specifically tailored for security threat data, addressing a critical gap in existing literature. Our detailed analysis of learning rate dynamics in LLM models revealed previously undocumented patterns specific to security applications. We created a distinctive analytical approach that uniquely considered both structural and contextual aspects of code, enabling detection of sophisticated threats that evaded traditional methods. Our high-accuracy implementation demonstrated unprecedented performance in real-world security environments. Our work established a new research direction by extending these methodologies beyond cybersecurity to other domains requiring integrated text and code analysis. The foundational frameworks we developed opened previously unexplored pathways for real-time security monitoring and customized threat detection solutions.

The rapidly evolving cybersecurity landscape revealed critical gaps in existing analytical approaches, particularly in connecting social media threat indicators with code vulnerabilities—a connection increasingly exploited by sophisticated attackers but overlooked in research. The persistent challenge of underrepresented threat types in security datasets created blind spots in existing detection systems. Our research directly addressed this fundamental issue that previous approaches failed to recognize or adequately solve. Conventional code vulnerability analysis methods consistently failed to scale with modern codebases or capture semantic relationships essential for detecting advanced threats. The complexity of modern attack patterns, particularly those involving discontinuous code elements, demanded fundamentally new approaches. The dichotomy between ML models (lacking contextual understanding) and LLMs (with prohibitive computational costs) represented a significant barrier to effective security threat detection that had not been systematically addressed in existing literature. The absence of multi-dimensional analytical frameworks integrating syntax, semantics, and information flow left a critical gap in security research that our work specifically targeted, enabling detection of complex threat patterns invisible to conventional single-dimension approaches.

Literature review

Hossain et al. apply machine learning and deep learning algorithms to detect signs of attack[10]. Research has limitations in addressing the problem of class imbalance. In a study by Tait et al.[11] machine learning techniques were applied to the classification of anomaly detection signatures, and binary classification showed the highest performance in Random Forest and multi-classification in KNN. However, the study may have insufficient

data on specific attack types, which may lead to class imbalance. In their study, Kocher and Kumar applied the stochastic gradient descent method, random forest, and naive Bayes to analyze the UNSW-NB15 dataset using machine learning algorithms[12]. They applied a chi-square-based feature selection technique to improve the performance of model detection. The dataset contains 49 features and 2,544,044 records. The Chi-square technique is applied to select 23 key features out of 49. However, the study has the disadvantage of being applied only to supervised learning. Chakrawarti and Shrivastava's research is to find various machine learning algorithms, including random forest and SVM, that have high performance in detecting signs of attacks[13]. The dataset used is focused on a specific type of attack and has limitations in model training in a heterogeneous dataset environment. It is necessary to consider the problem of data imbalance between normal and attack information in the dataset. Studies by Mahmood et al.[14] focus on feature selection and machine learning classifiers for detecting signs of data attacks. The applied classifiers include decision trees and SVMs. Feature selection applies genetic algorithms to optimal features and focuses on particle swarm optimization. XGBoost-based feature selection is excluded. In Huang and Aumpansub[15] research, a deep learning model is used to detect weaknesses in C/C++ code. Weaknesses in code fragments are extracted and cosine similarity-based Word2Vec embedding is applied. The data imbalance problem is balanced by down-sampling normal and abnormal code. It has limitations in determining only whether a vulnerability exists. In studies by Muniz and others[16] static analysis is performed to identify potential weaknesses in the code. Static analysis tools have the limitation of producing false positives due to a lack of full understanding of the context of the code. Studies by Almahmoud et al.[17] use the Hackmageddon dataset for air threat prediction. It contains 42 types of cyberattacks from 36 countries. The machine learning model uses Bayesian LSTM and takes uncertainty into account in Bayesian inference. However, there is a limitation in that other transformer-based models were excluded from the experiment. Studies such as Khandpur are using social media data to detect cyber- attack[18]. Post News data is used to expand queries and detect signs of attack by combining machine learning such as crowdsourcing, random forest, and SVM. Considering the bias of social media data, syntax analysis or additional measures are required. In a study by Zhou et al.[19] the focus is on detecting proactive aggression in offensive behavior on social media by applying a deep learning-based language model. The model used in the study is BERT, and its performance is compared with that of ML-based models such as SVM and CNN-LSTM. However, the paper states that the BERT model is based on a static dataset, so comparative studies with other models are needed. Chakraborty et al. evaluate deep learning models for detecting software vulnerabilities, comparing them with existing techniques and finding a need for deeper code analysis[20]. Zhou et al. use graph neural networks (GNN) for vulnerability detection, noting high computational demands and scalability issues[21]. Hin et al.[22] propose Linevd for statement-level vulnerability detection, lacking full code context integration. Li et al.[23] improve model predictions using various code characteristics but face performance degradation concerns. Cheng et al.[24] introduce a contrastive learning technique for path-sensitive code insertion, limited by data quality and computation needs. Ding et al.[25] show that language models can learn code patterns, though they require large datasets. Khare et al.[26] find LLMs effective in identifying security vulnerabilities but point out high costs and limitations in large-scale data training. Steenhoek et al.[27] show difficulties in detecting complex vulnerabilities with LLMs and suggest considering code grammar and semantic information. Xia et al.[28] study automatic program modification using pre-trained LLMs, facing challenges in correcting errors in complex structures. Joshi et al.[29] explore multilingual program modifications with LLMs, highlighting issues in providing correction explanations. Xia et al.[30] use LLMs for generating fuzzing test cases, emphasizing the need for balanced training data. Zhang et al.[31] present Autocoderover for autonomous bug fixes, requiring effective dataset building for LLM training. Mittal et al.[32] detect security threats using social media data but face inefficiencies in data filtering mechanisms. Li et al.[33] point out AST structural limitations, needing further research. Huo et al.[34] integrate static and dynamic analyses, limited by high computational loads. Wu et al.[35] propose integrated analysis models for various data types in text and code. Nayyef et al.[36] analyzed the limitations of algorithms such as regression analysis, time series forecasting, neural networks, and SVM and data cases in energy optimisation research for smart grids. In addition, Mensah et al.[37] analyzed the impact on the financial industry based on artificial intelligence and machine learning technologies.

This study aimed to effectively detect security threats by integrating text-based security news with source code vulnerability detection. To achieve this, we examined detection models employing machine learning (ML) techniques and large language models (LLMs), as well as models for identifying vulnerabilities in open-source code. The dataset comprised Post News data and C/C++ source code data. The Post News data was categorized into three classes: high-risk, general news, and non-news, utilizing the dataset provided by CybAttT. Data preprocessing involved tokenization, scaling, and normalization, specifically applying ratio-based scaling, logarithmic scaling, and normalization-based weighting to address bias issues. The C/C++ source code data was compiled using the NIST Juliet v1.3 vulnerability sample dataset and included information such as call functions, call function roles, and abstract syntax trees. During data preparation, we designed a dataset incorporating syntax analysis, semantic analysis, and information flow learning. The embedding process varied depending on whether an ML model or an LLM was used. For ML models, TF-IDF vectorization, Word2Vec vectorization, and label encoding were applied. For LLMs, specialized tokenizers were utilized for each model, and pre-trained model such as BERT, CodeBERT, GraphCodeBERT, ST5-xxl, and XLNet were employed. Hyperparameter tuning was conducted to optimize the model, deriving the optimal parameter combination through grid search and k-fold cross-validation. Throughout this process, we addressed dataset imbalance, constructed the dataset to reflect syntax, semantics, and information flow, and adjusted weights via scaling and normalization. An integrated approach was adopted, utilizing ML models such as XGBoost, SVM, and Random Forest along with various LLM models. The primary research objective was the effective detection of security threats by combining text-based security news with source code vulnerability detection. We aimed to develop detection models leveraging ML and LLM approaches and investigate methods for identifying weaknesses in open-source code. Our research

pioneered the first integrated approach to security threat detection by simultaneously analyzing social media data and source code vulnerabilities through a novel hybrid ML-LLM framework, addressing critical limitations where previous studies treated these domains separately. We developed a unique dual-analysis methodology bridging textual threat intelligence and code vulnerability detection that solved longstanding challenges in analyzing discontinuous code segments and handling severe class imbalance in security datasets. Our specialized weight optimization techniques achieved breakthrough performance with underrepresented threat classes, while our novel combination of AST extraction with code graph analysis achieved near-perfect accuracy rates in vulnerability detection. We established the first tri-dimensional security analysis framework integrating syntax, semantics, and information flow analysis, demonstrating that domain-specific approaches significantly outperformed general methods for detecting sophisticated cyber threats.

Method

Research methods

Figure 1 shows the work process of the proposed research. Build datasets that take into accounts the characteristics of each domain and utilize them as input information. Apply natural language processing techniques and AST-based analysis to process text and source code, respectively. Experiments are conducted on Post News data, which is text data, and C/C++ data, which is source code data. Post News data is raw data and biased data. The source data is the data that has been stripped of its convenience by taking scaling and normalization to be used as learning data. The source data was used for learning. C/C++ data is raw data, with weakness data provided by NIST's Juliet 1.3. The source data is the data optimized for use in ML and LLM models. Embedding technology suitable for the ML and LLM series has been applied. The learning models are ML and LLM. The ML series is trained with XGBoost, SVM, and Random Forest models, and the LLM series is trained with BERT, CodeBERT, GraphCodeBERT, ST5-xxl, and XLNet models. The evaluation metrics were applied to ML models and LLM. As a result, the results of the model with superior performance are determined.

The preprocessing stage of the dataset, especially the method of dealing with data imbalance, had been described and the specific parameter values and the rationale for their determination had been presented. In the ML model experiment, Post New data was unbalanced data, so data preprocessing had been performed based on the weight adjustment method (rate based, log scale, normalization based). When determining the optimal value for weight adjustment, the weight that provided the best performance in the given metric had been selected. C/C++ weakness code provided balanced data, so there was no need to process unbalanced data. Post New data and C/C++ code data both had applied TF-IDF/Word2Vec embedding.

Figure 2 shows the flow for data configuration data sets such as Post News, C/C++ weakness codes.

Data sets

The data set had been updated to include a description of 6(train):2(test):2(validation).

Post News data included 21,000 cyber tweets with an annotation indicating whether or not they were cyber-attacked (https://github.com/HudaLughbi/CybAttT). The data consisted of the text of the cyber security tweet, the results labelled by the annotators (annotator1, annotator2, annotator3), and the final label selected by multiple annotators (majority). There were three labelled pieces of information: 'high-risk news', 'normal news', and 'not news'. The final label was determined by majority voting. The total number of data was 36,071 Post News, and

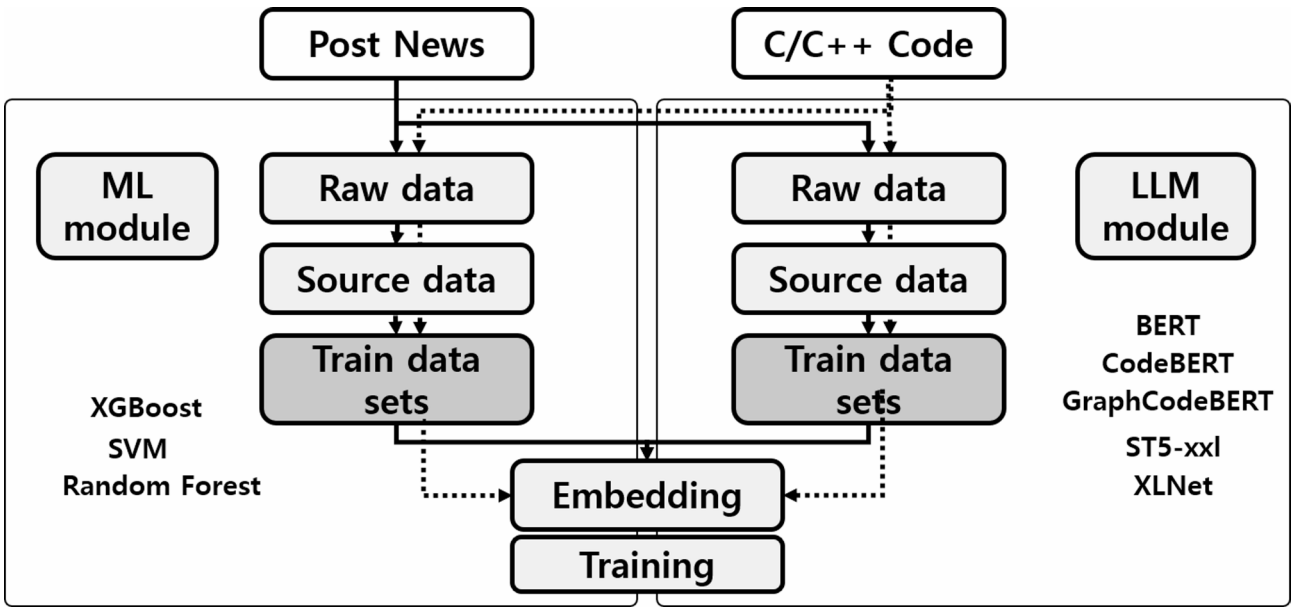

Fig. 1. The work process of the proposed research.

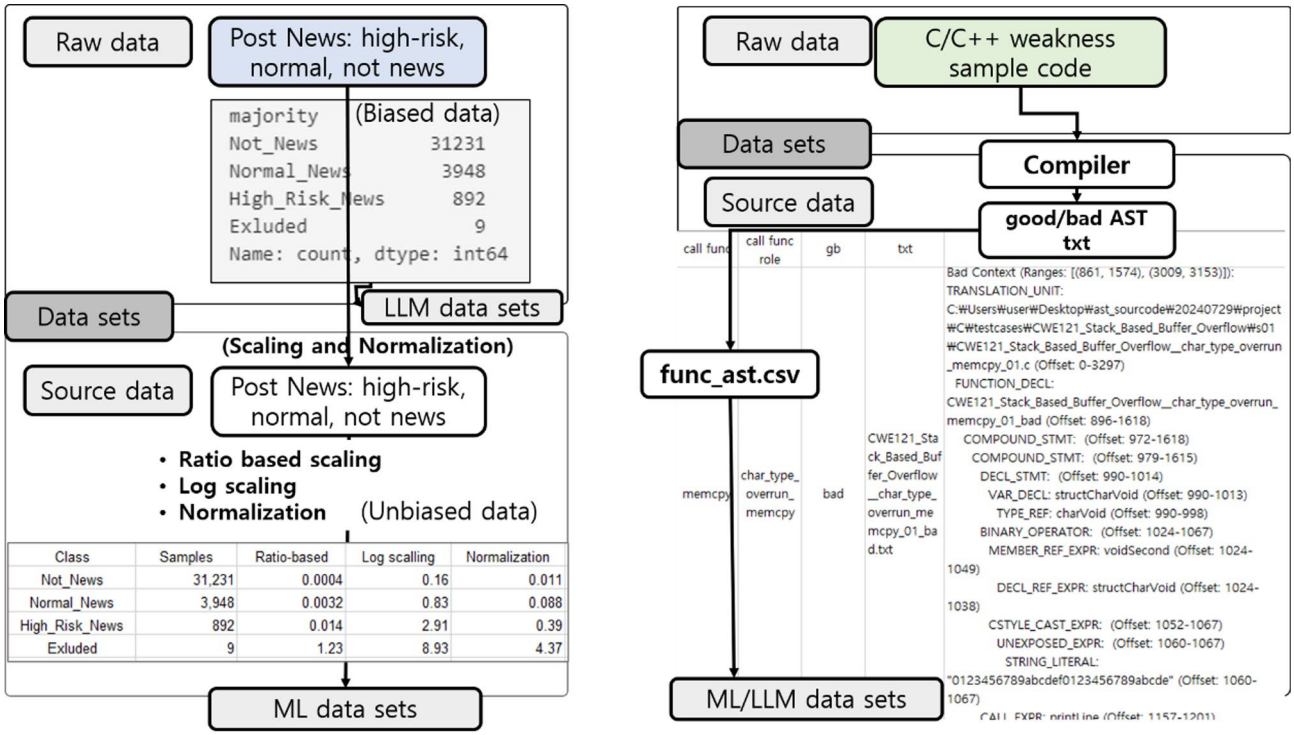

Fig. 2. Flow for configuration the data sets (Post News; C/C++ weakness codes).

each tweet contained up to 512 characters. In the pre-processing process, unnecessary characters were removed from the tweets and converted to lowercase. Then, tokenization and attention masks were generated.

NIST's Juliet v1.3 C/C++ weakness dataset contained types such as buffer overflow, memory leak, and format string vulnerabilities. It provided code examples, which were classified according to the common weakness enumeration. The code size was in the form of a function or a small program, and consisted of concise code. It was also stored in the form of an abstract syntax tree. Juliet v1.3 contained tens of thousands of test cases, each weakness was paired with a 'good' and a 'bad' code. Of course, the test suite contained more than 61,000 test cases across more than 100 CWE categories.

For C/C++ data, using the compiler, good/bad code had been extracted from the C/C++ code and stored in the good/bad column, and the call function had been stored in call_func. Information on what kind of call function it was stored in the call_func_role column. The corresponding code converted to AST had been generated as a txt file, so the txt column existed, and the AST that existed as a txt file had been stored in the AST column and organized.

This was a summary of the functions used for feature extraction. The functions used for feature extraction included text preprocessing functions that removed unnecessary characters, converted lowercase letters, and processed text or code with an attention mask for tokenization or batch processing, code analysis functions that extracted good/bad code through a compiler, stored call functions in the call_func column, stored call function type information, converted and stored AST, etc. And it consisted of feature weighting functions including ratio-based scaling, log scaling, and normalization-based weighting functions, as well as TF-IDF/Word2Vec embedding functions. Embedding functions converted text or code data into numerical vectors. A code structure analysis function was included, which converted source code into AST, extracted and classified call triples, and used a function to quantify the structural features of the code. Call_func extraction identified function calls in the source code and stored the identified function call names in the call_func column. call_func_role extraction classified the role or type of the called function, such as memory allocation or deallocation functions, and stored this information in the call_func_role column. Good/bad code classification identified good and bad codes based on information provided by the dataset and stored them in the good/bad column.

Post news data sets

The news data used for post news information-based threat signature detection was a dataset provided by CybAttT. Post News data is categorized into three classes: high-risk, normal news, and not news. The dataset of input X can be represented by the following equation.

$$X = X_1, X_2, \ldots, X_n \tag{1}$$

X_{i1} is the frequency count of high-risk, X_{i2} is the frequency count of normal, and X_{i3} is the frequency count of not news.

The dataset of the output y can be represented by the following Eq. (1).

$$y = y_1, y_2, \ldots, y_n \tag{2}$$

Data preprocessing includes tokenization, scaling, and normalization. Here, tokenization can be represented by the following Eq. (3).

$$T(x_i) = \{t_{i1}, t_{i2}, \ldots, t_{im}\}, \forall x_i \in X \tag{3}$$

$T(x_i)$ is the set of tokens obtained by tokenizing data x_i. Let t_{i1} be the individual tokens generated from xi, and each token can be represented as a word, partial word, character, etc. m is the number of tokens and is determined by the length of the data and the tokenizer. ∀xi∈X performs the tokenization for each data point xi in the entire dataset X.

Optimization on the Post News dataset deals with the problem of constructing unbiased data sets by scaling and normalization due to the biased nature of the dataset. In this study, scaling and normalization are used to solve the bias, and the methods include ratio based 52 scaling, log scaling, and normalization-based weighting.

Ratio based scaling

Ratio-based scaling weighting is used to express the relative importance of data within a set and follows Eq. (4) to calculate the relative ratio. The sum of the values is always 1. Consider the ratio of the value of data x_{ij} to the values within the same data point $x_{ij} = [x_{i1}, x_{i2}, x_{i3}]$.

$$x'_{ij} = \frac{x_{ij}}{\sum_{k=1}^{m} x_{ik}}, \forall i, j \tag{4}$$

Here, the value of data x_{ij} is divided by the sum of the values in x_i. x'_{ij} has a value in [0,1].

Weight based log scaling

Log-scale weighting normalizes asymmetric data. It is effective when the differences in values are extremely large. When the values in the data are large or asymmetrically distributed, the logarithmic transformation scales, distributes, and normalizes them, following Eq. (5).

$$x'_{ij} = \ln(x_{ij} + 1), \forall i, j \tag{5}$$

Here we add 1 to x_{ij} and take *ln*. We are adding + 1 to make it definable if x_{ij} is 0.

Weight based normalization

The expression to normalize all data points to the same range follows Eq. (6) and gives the data x_{ij} a value in the range [0,1] based on the minimum and maximum values of the corresponding data points $x_i = [x_{i1}, x_{i2}, x_{i3}]$.

$$x'_{ij} = \frac{x_{ij} - min(x_i)}{max(x_i) - min(x_i)}, \forall i, j \tag{6}$$

In Eq. (6), we subtract the minimum value min (x_i) from x_{ij}, and divide by the entire range (max(x_i)—min(x_i)).

Another dataset used in our experiments is the sample data set of C/C++ source code weaknesses provided by NIST Juliet v1.3.

C/C++ source code-based weakness data sets

Similarly, the code sample dataset with C/C++ source code weaknesses consists of X and y. For the source code data set, Eqs. (1) through (4) can be applied to the input X and output y, even though the data contents are different.

C/C++ source code input information X_{i1} is represented by call func and is a call function. X_{i2} is a call func role, and X_{i3} is an abstract structure tree. Of course, for y_i∈{g, b}, the output y means the result value of good if g and bad if b.

Referring to Eq. (7), the input xi is tokenized by the model tokenizer T, which can be expressed as Eq. (6).

$$Xi \rightarrow T(Xi) = \{t_{i1}, t_{i2}, \ldots, t_{im}\} \tag{7}$$

The output $T(X_i)$, when constructed as a set of individual tokens, is represented as in Eq. (8).

$$T(X_i) = \{\text{call func, call func role, AST}\} \tag{8}$$

In this study, when constructing the C/C++ source code weakness sample dataset, we designed it to include not only syntactic analysis of the weakness sample code in the source code, but also semantic analysis and information flow learning.

Figure 3 introduces the embedding functions used in ML models and LLM, the data sets used in learning, the parameters for each learning model, and the parameters used as evaluation indicators.

ML embedding setting for post news data

TF-IDF vectorization

$$X = TF - IDF_{10000}\left(\left(data['text']\right)\right) \tag{9}$$

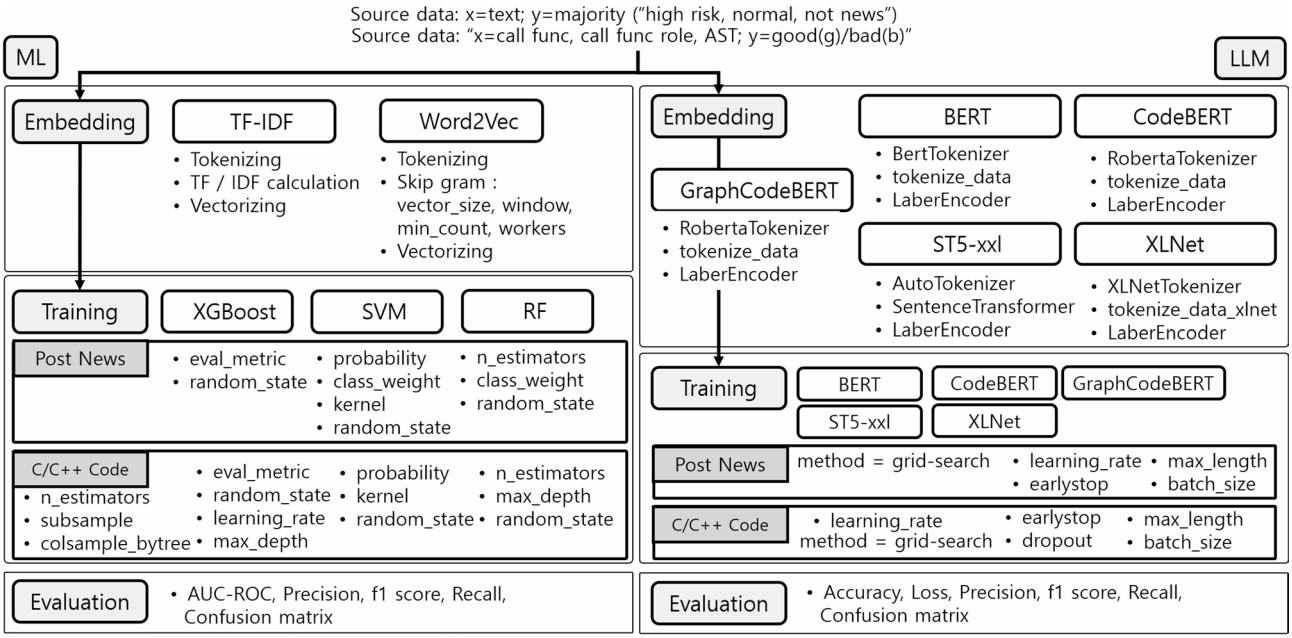

Fig. 3. Tuning parameters used in step of embedding, training, and evaluation stages.

Extract Labels

$$y = data\left(\left['majority'\right]\right) \tag{10}$$

Label Encoding

$$y_{encoded} = LabelEncoder().fit\_transform\,(y) \tag{11}$$

Word2Vec vectorization
Convert Text Data to List

$$texts = data['text'].astype\,(str)\,.tolist() \tag{12}$$

Extracted labels

$$labels = data['majority'] \tag{13}$$

Train Word2Vec Model

$$\begin{gathered} Word2Vec\_model = Word2Vec\,(sentences\,[text.split()\,fortextintexts] \\ vector\_size = 100, window = 5, \min\_count = 1, workers = 4) \end{gathered} \tag{14}$$

Convert text data to Word2Vec embeddings

$$X = np.array([embed\_sentence\,(text, word2vec\_model, vector\_size]\,fortextintexts) \tag{15}$$

Label encoding

$$y_{encoded} = LabelEncoder().fit\_transdform\,(labels) \tag{16}$$

ML embedding setting for C/C++ source code-based weakness data

TF-IDF/Word2Vec vectorization

$$x_{call\_function} = TF - IDF_{1000}\left(df\left['call\_function'\right]\right) \tag{17}$$

$$x_{call\_function} = TF - IDF_{1000}\left(df['call\_function\_role'\right]) \tag{18}$$

$$x_{ast} = TF - IDF_{5000}\left(df\left['AST'\right]\right) \tag{19}$$

Input Vector Creation

$$X = hstack\left(\left[X_{call\_func}, X_{call\_func\_role}, X_{AST}\right]\right) \tag{20}$$

Label Encoding

$$y_{g/b} = LabelEncoder().fit\_transform\left(df\left['g/b'\right]\right) \tag{21}$$

In our integrated study, we have various approaches to solve the problem of dataset imbalance, construct datasets that reflect the syntax, semantics, and information flow of the data, select hyperparameters, process weights using scaling and normalization techniques, process embeddings using TF-IDF and Word2Vec, and train models using ML models such as XGBoost, SVM, and Random Forest, and LLM models such as BERT, CodeBERT, GraphCodeBERT, ST5-xxl, and XLNet.

LLM embedding setting for post news data

Data Tokenization

$$input\_ids, attention\_masks = tokenize\_data\left(data\left['text'\right]\right) \tag{22}$$

Label encoding

$$encoded\_labels = label\_encoder.fit\_transform\left(data\left['majority'\right]\right) \tag{23}$$

Number of label classes

$$num\_labels = |LabelEncoder().classes| \tag{24}$$

This code segment shows the process of tokenizing text data and encoding labels for a machine learning model. It performs data tokenization on input text, applies label encoding to the 'majority' field, and retrieves the total number of unique label classes.

LLM embedding setting for C/C++ source code-based weakness data

Tokenizing each column

$$call\_func\_ids, call\_func\_masks = tokenize\ column\left(df\left['call\_func'\right]\right) \tag{25}$$

$$call\_func\_role\_ids, call\_func\_role\_masks = tokenize\,column\left(df\left['call\_func\_role'\right]\right) \tag{26}$$

$$ast\_ids, ast\_masks = tokenizecolumn\left(df\left['AST'\right]\right) \tag{27}$$

Merging input data

$$input\_ids = concat\left([call\_func\_ids, call\_func\_role\_ids, ast\_ids], dim = 1\right. \tag{28}$$

$$attention\_masks = concat\left(\left[call_{func_{masks}}, call_{func_{role_{masks}}}, ast_{masks}\right], dim = 1\right. \tag{29}$$

Label encoding

$$labels = LabelEncoder(), fittransform\left(df['g/b'\right]) \tag{30}$$

This code snippet shows a data processing pipeline for natural language processing, specifically tokenizing different columns of input data (call functions, roles, and AST). The process involves tokenizing each column separately, merging the tokenized data, and then encoding the context labels using a LabelEncoder.

LLM tokenizer and cross validation

The tokenizer and cross-validation process of the models used in this study are shown in Table 2 below. First, different pre-trained torquemeters were applied in the embedding stage for each model. For the BERT model, the BertTokenizer based on 'bert-base-uncased' was used, and for CodeBERT, the RobertaTokenizer based on 'microsoft/codebert-base' was used. GraphCodeBERT applied the RobertaTokenizer based on 'microsoft/graphcodebert-base', ST5-xxl applied the T5Tokenizer based on 'google/t5-xxl', and XLNet applied the XLNetTokenizer based on 'xlnet-large-cased'. In the learning phase, various hyperparameters were considered to optimize the performance of the LLM models. The main hyperparameters include learning_rate, earlystop, max_length, dropout, and batch_size. To find the optimal combination of these parameters, we performed a grid search and k-fold cross-validation. The verification loss (Lcv) for evaluating the performance of the model is calculated as the average of the verification losses (L(j)) obtained from k folds, and the optimal parameter combination is derived through this. Through this systematic model learning and verification process, the performance of each model was optimized Table 3.

Workflow	Model	Tokenizer of models and cross validation
Embedding	BERT	tokenizer = BertTokenizer.from_pretrained("bert-base-uncased") For input x,$tokens = BertTokenizer(x)$
	CodeBERT	tokenizer = RobertaTokenizer.from_pretrained("microsoft/codebert-base") For input x,$tokens = RobertaTokenizer(x)$
	Graph CodeBERT	tokenizer = RobertaTokenizer.from_pretrained("microsoft/graphcodebert-base") For input x,$tokens = RobertaTokenizer(x)$
	ST5-xxl	tokenizer = T5Tokenizer.from_pretrained("google/t5-xxl") For input x,$tokens = AutoTokenizer(x)$
	XLNet	tokenizer = XLNetTokenizer.from_pretrained("xlnet-large-cased") For input x,$tokens = XLNetTokenizer(x)$
Training	LLM Models	Important hyperparameters in the LLM Models training process are learning_rate, earlystop, max_length, dropout, batch_size, method = grid-search, and K-fold Cross validation (CV). $L_{cv}(h) = \frac{1}{k}\sum_{j=1}^{k} L^{(j)}(h)$ $L^{(j)}(h)$ is the verification loss at the j th fold The optimal combination of parameters follows the following equation $h^* = arg_{h_i \in \mathrm{H}}{}^{min} L_{cv}(h)$

Table 2. Tokenizer and cross validation embedding and training of model.

Model	Learning rate (lr)	Epochs	Dropout_rate	Batch size	Optimizer
BERT	3.3e−5 ~ 3.9e−5	15 ~ 20	0.4 ~ 0.42	32	Adam
CodeBERT	2.5e−5 ~ 3.2e−5	15 ~ 20	0.36 ~ 0.4	32	
GraphCodeBERT	3.5e−5 ~ 4.1e−5	15 ~ 20	0.36 ~ 0.4	32	
XLNet	2e−5 ~ 3e−5	15 ~ 20	0.36 ~ 0.4	32	
ST5-xxl	1e−5 ~ 3e−5	15 ~ 20	0.38 ~ 0.4	32	

Table 3. Optimal tuning values of LLM parameters using Post News data.

Model	Learning rate (lr)	Epochs	Dropout_rate	Batch size	Optimizer
BERT	3.3e−5 ~ 3.9e−5	10 ~ 15	0.4 ~ 0.42	16	Adam
CodeBERT	2.5e−5 ~ 3.2e−5	5 ~ 10	0.36 ~ 0.4	16	
GraphCodeBERT	3.5e−5 ~ 4.1e−5	5 ~ 10	0.36 ~ 0.4	16	
XLNet	2.6e−5 ~ 2.7e−5	10 ~ 15	0.36 ~ 0.4	16	
ST5-xxl	2.2e−5 ~ 2.4e−5	10 ~ 12	0.38 ~ 0.4	16	

Table 4. Optimal tuning values of LLM parameters using Juliet C/C++ weakness code.

Results and discussions

f1 score experiment results of post news data

Optimal tuning values of ML parameters using post news data

When experimenting with the Post News data, the hyperparameter tuning range for optimal performance of the LLM models is as follows. The BERT model performed best at a learning rate between 3.3e−5 and 3.9e−5, with the dropout ratio optimized in the range of 0.4 to 0.42. For CodeBERT, the best results were obtained with a learning rate between 2.5e−5 and 3.2e−5 and a dropout ratio of 0.36 to 0.4. GraphCodeBERT showed optimal performance at a learning rate between 3.5e−5 and 4.1e−5 and a dropout ratio of 0.36 to 0.4. XLNet was most effective in the learning rate range between 2e−5 and 3e−5, and the dropout ratio of 0.36 to 0.4 was found to be optimal. ST5-xxl achieved the best performance at a relatively low learning rate between 1e−5 and 3e−5 and a dropout ratio of 0.38 to 0.4. For all models, the optimal results were obtained when the epoch was set to 15–20 and the batch size was set to 32, and the model was optimized using the Adam optimizer.

Optimal tuning values of LLM parameters using Juliet C/C++ weakness code

When the weakness code of C/C++ was targeted for the experiment, the optimal hyperparameter tuning range of the LLM models is as shown in Table 4 below. The BERT model performed best with a learning rate between 3.3e−5 and 3.9e−5, a dropout ratio between 0.4 and 0.42, and epochs between 10 and 15. CodeBERT achieved the best results with a learning rate between 2.5e−5 and 3.2e−5, a dropout ratio between 0.36 and 0.4, and epochs between 5 and 10. For GraphCodeBERT, the optimal performance was achieved with a learning rate between 3.5e−5 and 4.1e−5, a dropout ratio between 0.36 and 0.4, and epochs between 5 and 10. XLNet performed best with a learning rate between 2.6e−5 and 2.7e−5, a dropout ratio between 0.36 and 0.4, and epochs between 5 and 10. ST5-xxl achieved its best performance at a learning rate between 2.2e−5 and 2.4e−5, a dropout ratio between

0.38 and 0.4, and epochs between 10 and 12. For all models, the batch size was set to 16, and the Adam optimizer was used to optimize the models.

f1 score results of ML model using post news data

In Table 5, the performance evaluation results of ML models are as follows. In the case of the XGBoost model, the best performance was achieved with a macro f1 score of 0.66, a weighted average f1 score of 0.96, and an AUC-ROC of 0.9223 when TF-IDF embedding and ratio-based weighting were applied. When using Word2Vec embedding, the overall performance tended to decrease somewhat. The SVM model showed similar performance (macro f1 0.66, weighted average f1 0.97) when TF-IDF embedding was used and when the ratio, log scale, and normalization weight were applied. When Word2Vec embedding was applied, performance was somewhat reduced. In the case of the Random Forest model, there was relatively little difference in performance depending on the embedding method and weighting method. When TF-IDF embedding was used, the macro F1 score was 0.65, the weighted average F1 score was 0.96, and the AUC-ROC was 0.86, showing stable performance. Overall, TF-IDF embedding performed better than Word2Vec, and the ratio-based method was the most effective among the weighting methods.

f1 score results of LLM using post news data

The f1 scores that evaluated the performance of each LLM model tested in this study are shown in Table 6 below. The ST5-xxl model recorded a macro F1 score of 0.6657 and a micro F1 score of 0.9728 at learning rate lr2e−5, and a macro of 0.6672 and a micro of 0.9725 at lr2.5e−5. XLNet achieved a macro of 0.6540 and a micro of 0.9686 at a learning rate of lr2e−5, a macro of 0.6559 and a micro of 0.9686 at lr2.25e−5, and a macro of 0.6538 and a micro of 0.9695 at lr3e−5.

BERT achieved a macro of 0.6624 and a micro of 0.9714 at a learning rate of lr2e−5, a macro of 0.6661 and a micro of 0.9725 at lr3e−5, and a macro of 0.6504 and a micro of 0.9686 at lr4e−5. CodeBERT showed the following results at a learning rate of 2.68e−5: macro 0.6556, micro 0.9689; at 2.85e−5: macro 0.6563, micro 0.9711; at lr3.2e−5: macro 0.6583, micro 0.9703. GraphCodeBERT recorded the following performance at the learning rate of lr3.5e−5: macro 0.6563, micro 0.9711; at lr3.8e−5: macro 0.6538, micro 0.9692; and at lr3.95e−5: macro 0.6547, micro 0.9684. Overall, the ST5-xxl model performed the best compared to the other models, achieving high performance in the micro f1 score in particular.

Accuracy experiment results of C/C++ weakness code data

The results of the accuracy evaluation of the LLM and ML models in this study are as follows in Table 7. Looking at the performance of the LLM models, ST5-xxl achieved a very high accuracy of 0.999999999 with a learning rate of 2.4e−5 and a dropout of 0.385. XLNet showed an accuracy of 0.999999996 at a learning rate of 2.6e−5 and a dropout of 0.36. BERT showed an accuracy of 0.903703 at a learning rate of 3.45e−5 and a dropout of 0.4375, while CodeBERT achieved a high accuracy of 0.99999894 at a learning rate of 2.5e−5 and a dropout of 0.4. GraphCodeBERT showed excellent performance of 0.99999999 at a learning rate of 4.25e−5 and dropout of 0.39.

For ML models, XGBoost showed a perfect accuracy of 1.0000 when using TF-IDF embedding, and an accuracy of 0.9002 when using Word2Vec embedding. SVM achieved an accuracy of 0.9699 in TF-IDF embedding and 0.8945 in Word2Vec embedding. Random Forest showed performance of 0.9493 in TF-IDF embedding and 0.8826 in Word2Vec embedding. Overall, the LLM models showed very high accuracy, and

Model	Embedding	Weight	f1 score (macro avg)	f1 score (weighted avg)	AUC-ROC
XGBoost	TF-IDF	Ratio	0.66	0.96	0.9223
		Log scale	0.64	0.95	0.8907
		Normalization	0.44	0.80	0.8281
	Word2Vec	ratio	0.63	0.94	0.9511
		Log scale	0.62	0.92	0.9294
		Normalization	0.48	0.85	0.8325
SVM	TF-IDF	Ratio	0.66	0.97	
		Log scale	0.66	0.97	
		Normalization	0.66	0.97	
	Word2Vec	Ratio	0.48	0.85	
		Log scale	0.60	0.89	
		Normalization	0.48	0.85	
Random Forest	TF-IDF	Ratio	0.65	0.96	0.8646
		Log scale	0.65	0.96	0.8639
		Normalization	0.65	0.96	0.8639
	Word2Vec	Ratio	0.60	0.94	0.8446
		Log scale	0.61	0.94	0.8463
		Normalization	0.60	0.94	0.8467

Table 5. f1 score results of ML models in Post News data.

Model		f1 score	
		Macro	Micro
ST5-xxl	lr2e−5	0.665797509276686	0.972838137472283
	lr2.5e−5	0.667245615588264	0.972560975609756
XLNet	lr2e−5	0.654068391677913	0.968680709534368
	lr2.25e−5	0.655954967594391	0.968680709534368
	lr3e−5	0.65384566426361	0.969512195121951
BERT	lr2e−5	0.66242	0.97145
	lr3e−5	0.66612	0.97256
	lr4e−5	0.65046	0.96868
CodeBERT	2.68e−5	0.65569	0.96896
	2.85e−5	0.65633	0.97118
	lr3.2e−5	0.65837	0.97034
GraphCodeBERT	lr3.5e−5	0.65633	0.97118
	lr3.8e−5	0.65388	0.96924
	lr3.95e−5	0.65477	0.9684

Table 6. f1 score experiment results of LLM.

Type	Model	Embedding	Ir	dropout	Mean Accuracy	Confidence interval (95%)	Error margin
LLM	ST5-xxl	–	2.4e−5	0.385	0.99999999999149	(lower) 0.9999999995 (upper) 1.0000000000	2.43188247317505E−10
	XLNet	–	2.6e−5	0.36	0.9999999996	(lower) 0.3499604671 (upper) 0.9934291045	0.321734318709874
	BERT	–	3.45e−5	0.4375	0.903703	(lower) 0.8850202144 (upper) 0.9037031188	0.00934145217911064
	CodeBERT	–	2.5e−5	0.4	0.999999894	(lower) 0.9999998632 (upper) 0.9999999867	6.17414579462405E−08
	Graph CodeBERT	–	4.25e−5	0.39	0.999999999991	(lower) 0.9999966838 (upper) 1.0000010721	2.19417855784609E−06
ML	XGBoost	TF-IDF			≈1.0000	(lower) 1.0000 (upper) 1.0000	0.0000
		Word2Vec			0.9002	(lower) 0.8812 (upper) 0.9192	0.0190
	SVM	TF-IDF			0.9699	(lower) 0.9574 (upper) 0.9824	0.0125
		Word2Vec			0.8945	(lower) 0.8750 (upper) 0.9140	0.0195
	Random Forest	TF-IDF			0.9493	(lower) 0.9351 (upper) 0.9635	0.0142
		Word2Vec			0.8826	(lower) 0.8611 (upper) 0.9041	0.0215

Table 7. Accuracy experiment results of LLMs and ML models.

in the ML models, TF-IDF embedding performed better than Word2Vec. In particular, the combination of XGBoost and TF-IDF embedding achieved the highest accuracy.

Conclusion

Based on the experimental results of this study, several major discussion points can be derived. First, in the analysis of Post News data, the ML and LLM models demonstrated performance differences according to their respective characteristics. Among the ML models, XGBoost achieved excellent performance with a macro F1 score of 0.66, a weighted average F1 score of 0.96, and an AUC-ROC of 0.9223 when TF-IDF embedding and ratio-based weighting were applied. This indicates that the ensemble learning characteristics of XGBoost are effective in handling unbalanced data, particularly with the application of ratio-based weighting to address class imbalance issues. The SVM and Random Forest models also exhibited stable performance with TF-IDF embedding but showed a slight decrease in performance with Word2Vec embedding. In the LLM models, ST5-xxl demonstrated the highest performance, especially with a learning rate of 2.4e−5 and a dropout rate

of 0.385. This highlights the effectiveness of large−scale pre-training models' contextual understanding in detecting security threats. Additionally, other LLM models such as BERT, CodeBERT, and GraphCodeBERT also demonstrated high performance, suggesting that the specialized architecture of each model is suitable for text-based security threat detection.

Second, in the analysis of C/C++ code vulnerabilities, an approach integrating syntax analysis, semantic analysis, and information flow proved to be highly effective. Most LLM models achieved exceptionally high accuracy rates of 0.99 or higher, demonstrating the strength of pre-trained models in simultaneously considering the structural characteristics and context of code. Notably, ST5-xxl and GraphCodeBERT achieved near-perfect accuracy rates of 0.999 or higher. It is also significant that the combination of XGBoost and TF-IDF embedding achieved an accuracy rate of approximately 1.0000 in the ML models. This indicates that when appropriate feature extraction and model selection are combined, excellent performance in code vulnerability detection can be achieved.

Third, the choice of data preprocessing and embedding methods had a decisive impact on model performance. Overall, TF-IDF embedding outperformed Word2Vec, suggesting that the TF-IDF method, which considers word frequency and importance, is more effective for security-related text. Furthermore, the elimination of data bias through scaling and normalization significantly contributed to performance improvement. Ratio-based scaling was particularly effective, likely because it adequately reflects the relative importance of data.

Fourth, hyperparameter tuning played a crucial role in optimizing model performance. In LLM models, fine-tuning the learning rate and dropout ratio significantly impacted performance. For instance, in the ST5-xxl model, there was a noticeable performance difference even with a slight adjustment in the learning rate from 2.4e−5 to 2.3e−5. This indicates that the more complex the model, the more detailed parameter adjustments required, and that systematic hyperparameter search is essential for optimal performance. The choice of batch size and number of epochs was also an important factor. The optimal batch size was 32 for Post News data and 16 for C/C++ code analysis, suggesting that the appropriate batch size may vary depending on the characteristics and complexity of the data.

The ST5-xxl model was superior to BERT-type models. This was because it embedded sentences by sentence, which allowed for a high degree of contextual understanding and faster computation due to lower memory usage. The ST5-xxl model showed the most stable and fast convergence results compared to other models. We looked at the potential limitations and scenarios in which the ST5-xxl model might fail. The ST5-xxl model was a large pre-trained model, so it had a high computational load. A learning method for lightweight models was required. In addition, when processing long sentences, the length of tokens might increase, which could increase computational costs. This required excessive GPU memory usage, so it was necessary to design it to reduce memory usage.

The results of this study suggest the potential for effectively integrating ML and LLM models for security threat detection. It demonstrates the importance of selecting the appropriate model and preprocessing technique based on data characteristics and performing detailed hyperparameter tuning. These findings are expected to serve as important guidelines for developing real-time security monitoring systems in the future. Additionally, the methodology of this study is anticipated to be applicable to the analysis of texts and codes in other domains.

Data availability

The datasets used and/or analysed during the current study available from the corresponding author on reasonable request.

Received: 31 January 2025; Accepted: 30 May 2025
Published online: 02 July 2025

References

1. https://www.ibm.com/reports/data-breach
2. https://www.crowdstrike.com/en-us/global-threat-report/
3. Tripathy, S. S. & Behera, B. A review of various datasets for machine learning algorithm-based intrusion detection system: Advances and challenges. *Int. J. Intell. Syst. Appl. Eng.* **12**(4), 3833–3857 (2024).
4. Adeniyi, J. K. et al. EASESUM: an online abstractive and extractive text summarizer using deep learning technique. *Int. J. Artif. Intell.* **13**(2), 1888–1899. https://doi.org/10.11591/ijai.v13.i2.pp1888-1899 (2024).
5. Ajagbe, S. A. Developing Nigeria multilingual languages speech datasets for antenatal orientation. In *Applied Informatics. ICAI 2024. Communications in Computer and Information Science* Vol. 2237 (eds Florez, H. & Astudillo, H.) (Springer, 2025). https://doi.org/10.1007/978-3-031-75147-9_11.
6. Thakkar, A. & Lohiya, R. A review of the advancement in intrusion detection datasets. *Procedia Comput. Sci.* **167**, 636–645. https://doi.org/10.1016/j.procs.2020.03.330 (2020).
7. Zhou, Y., Cheng, G., Jiang, S. & Dai, M. Building an efficient intrusion detection system based on feature selection and ensemble classifier. *Comput. Netw.* **174**, 107247. https://doi.org/10.1016/j.comnet.2020.107247 (2020).
8. Maniriho, P. et al. Detecting intrusions in computer network traffic with machine learning approaches. *Int. J. Intell. Eng. Syst.* **13**(3), 433–445. https://doi.org/10.22266/ijies2020.0630.39 (2020).
9. Yilmaz, A. A. Intrusion Detection in Computer Networks using Optimized Machine Learning Algorithms. In *3rd International Informatics and Software Engineering Conference (IISEC)*. 1–5 (2022). https://doi.org/10.1109/IISEC56263.2022.9998258
10. Hossain, M. S., Goose, D., Partho, A. M., Ahmed, M., Chowdhury, M. T., Hasan, M. et al. Performance Evaluation of Intrusion Detection System Using Machine Learning and Deep Learning Algorithms. In *IBDAP*. 1–6(2023). https://doi.org/10.1109/IBDAP58581.2023.10271964.
11. Tait, K. -A., Khan, J. S., Alqahtani, F., Shah, A. A., Khan, F. A. et al. Intrusion Detection using Machine Learning Techniques: An Experimental Comparison. In *International Congress of Advanced Technology and Engineering (ICOTEN)*, 1–10 (2021). https://doi.org/10.1109/ICOTEN52080.2021.9493543.
12. Kocher, G. & Kumar, G. Analysis of ML algorithms with feature selection for intrusion detection using unsw-Nb15 dataset. *Int. J. Netw. Secur. Appl.* **13**(1), 21–31. https://doi.org/10.5121/ijnsa.2021.13102 (2021).

13. Chakrawarti, A. & Shrivastava, S. S. Enhancing intrusion detection system using deep q network approaches based on reinforcement learning. *Int. J. Intell. Syst. Appl. Eng.* **12**(12s), 34–45 (2024).
14. Mahmood, R. A. R., Abdi, A. & Hussin, M. Performance evaluation of intrusion detection system using selected features and machine learning classifiers. *Baghdad Sci. J.* **18**(2(Suppl.)), 0884. https://doi.org/10.21123/bsj.2021.18.2(Suppl.).0884 (2021).
15. Huang, Z., Aumpansub, A. Vulnerability Detection in C/C++ Code with Deep Learning. https://doi.org/10.48550/arXiv.2405.12384
16. Muniz, R., Andrade, W. & Machado, P. Towards a technique to detect weaknesses in C programs. In *SBES '21: Proceedings of the XXXV Brazilian Symposium on Software Engineering*. 39–48 (2021). https://doi.org/10.1145/3474624.3474633.
17. Almahmoud, Z., Yoo, P. D., Alhussein, M., Farhat, L. & Damiani, E. holistic and proactive approach to forecasting cyber threats. *Sci. Rep.* https://doi.org/10.1038/s41598-023-35198-1 (2023).
18. Khandpur, R. P., Ji, T., Jan, T., Jan, S., Wang, G., Lu, C.-T. & Ramakrishnan, N. Crowdsourcing cybersecurity: Cyber attack detection using social media. In *CIKM'17*. 1049–1057 (2017). https://doi.org/10.1145/3132847.3132866.
19. Zhou, Z., Yu, M., He, Y. & Peng, X. When cyber aggression prediction meets BERT on social media. 1–11 (2023). https://doi.org/10.48550/arXiv.2301.01877.
20. Chakraborty, S., Krishna, R., Ding, Y. & Ray, B. Deep learning-based vulnerability detection: Are we there yet. *IEEE Trans. Softw. Eng.* **48**, 3280–3296 (2020).
21. Zhou, Y., Liu, S., Siow, J., Du, X. & Liu, Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Neural Inform. Process. Syst.* https://doi.org/10.48550/arXiv.1909.03496 (2019).
22. Hin, D., Kan, A., Chen, H., and Babar, M. A. Linevd: Statement-level vulnerability detection using graph neural networks. In *IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 596–607 (2022). https://doi.org/10.48550/arXiv.2203.05181
23. Li, Y., Wang, S., and Nguyen, T. N. Vulnerability detection with fine-grained interpretations. In *The 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 292–303 (2021). https://doi.org/10.1145/3468264.3468597.
24. Cheng, X., Zhang, G., Wang, H., and Sui, Y. Path-sensitive code embedding via contrastive learning for software vulnerability detection. In *The 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 519–531(2022). https://doi.org/10.1145/3533767.3534371.
25. Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D., Ray, B. & Chen, Y. Vulnerability detection with code language models: How far are we?. https://doi.org/10.48550/arXiv.2403.18624 (2024).
26. Khare, S., Dutta, S., Li, Z., Solko-Breslin, A., Alur, R., and Naik, M. Understanding the effectiveness of large language models in detecting security vulnerabilities. https://doi.org/10.48550/arXiv.2311.16169 (2023).
27. Steenhoek, B., Rahman, M. M., Roy, M. K., Alam, M. S., Barr, E. T. & Le, W. To Err is Machine: Vulnerability Detection Challenges LLM Reasoning. https://doi.org/10.48550/arXiv.2403.17218 (2024).
28. Xia, C. S., Wei, Y., and Zhang, L. Automated program repair in the era of large pre-trained language models. In *The 45th International Conference on Software Engineering (ICSE 2023)*, 1482–1494(2023). https://doi.org/10.1109/ICSE48619.2023.00129.
29. Joshi, H. et al. Repair is nearly generation: Multilingual program repair with llms. *The AAAI Conf. Artif. Intell.* **37**, 5131–5140. https://doi.org/10.48550/arXiv.2208.11640 (2023).
30. Xia, C. S., Paltenghi, M., Le Tian, J., Pradel, M., and Zhang, L. Fuzz4all: Universal fuzzing with large language models. In *The IEEE/ACM 46th International Conference on Software Engineering*, 1–13 (2024). https://doi.org/10.1145/3597503.3639121.
31. Zhang, Y., Ruan, H., Fan, Z. & Roychoudhury, A. Autocoderover: Autonomous program improvement. In *The 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 1592–1604 (2024). https://doi.org/10.48550/arXiv.2404.05427.
32. Mittal, S., Das, P. K., Mulwad, V., Joshi, A. & Finin, T. CyberTwitter: Using Twitter to generate alerts for cybersecurity threats and vulnerabilities. In *The IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 860–867 (2016). https://doi.org/10.1109/ASONAM.2016.7752338.
33. Li, L., Feng, H., Zhuang, W., Meng, N. & Ryder, B. CClearner: a deep learning-based clone detection approach. In *The IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 249–260 (2017). https://doi.org/10.1109/ICSME.2017.46
34. Huo, X., Yang, Y., Li, M., and Zhan, D.-C. Learning semantic features for software defect prediction by code comments embedding. In *The IEEE International Conference on Data Mining (ICDM)*, 1049–1054 (2018). https://doi.org/10.1109/ICDM.2018.00133.
35. Wu, F., Wang, J., Liu, J., and Wang, W. Vulnerability detection with deep learning. In *The IEEE International Conference on Computer and Communications (ICCC)*, 1298–1302 (2017). https://doi.org/10.1109/CompComm.2017.8322752.
36. Nayyef, Z. T., Abdulrahman, M. M. & Kurdi, N. A. Optimizing energy efficiency in smart grids using machine learning algorithm a case study in electrical engineering. *SHIFRA* **2024**, 46–54. https://doi.org/10.70470/SHIFRA/2024/006 (2024).
37. Mensah, G. B. et al. The Era of AI: The impact of artificial (AI) and machine learning (ML) on financial stability in the banking sector. *EDRAAK* **2024**, 43–48. https://doi.org/10.70470/EDRAAK/2024/007 (2024).

Acknowledgements

This study is part of the master's thesis submitted by aSSIST University (Department of AI Big Data of graduate school) in Seoul, South Korea.

Author contributions

C.H. wrote the main manuscript text and T.O. supervised and edited the whole procedure of the research. All authors reviewed the manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to T.O.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.