systems

MDPI

Article

Enhancing Software Sustainability: Leveraging Large Language Models to Evaluate Security Requirements Fulfillment in Requirements Engineering

Ahmad F. Subahi

Department of Computer Science, University College of Al Jamoum, Umm Al-Qura University, Makkah 21421, Saudi Arabia; afsubahi@uqu.edu.sa

Abstract: In the digital era, cybersecurity is integral for preserving national security, digital privacy, and social sustainability. This research emphasizes the role of non-functional equirements (NFRs) in developing secure software systems that enhance societal wellbeing by ensuring data protection, user privacy, and system robustness. Specifically, this study introduces a proof-of-concept approach by leveraging machine learning (ML) models to classify NFRs and identify security-related issues early in the software development lifecycle. Two experiments were conducted to assess the effectiveness of different models for binary and multi-class classification tasks. In Experiment 1, BERT-based models and artificial neural networks (ANNs) were fine-tuned to classify NFRs into security and non-security categories using a dataset of 803 statements. BERT-based models outperformed ANNs, achieving higher accuracy, precision, recall, and ROC-AUC scores, with hyperparameter tuning further enhancing the results. Experiment 2 assessed logistic regression (LR), a support vector machine (SVM), and XGBoost for the multi-class classification of security-related NFRs into seven categories. The SVM and XGBoost showed strong performance, achieving high precision and recall in specific categories. The findings demonstrate the effectiveness of advanced ML models in automating NFR classification, improving software security, and supporting social sustainability. Future work will explore hybrid approaches to enhance scalability and accuracy.

Keywords: requirements engineering; SVM; large language models; BERT; security; non-functional requirements classification; data augmentation; software engineering

check for updates

Academic Editor: Vladimír Bureš

Received: 10 January 2025
Revised: 8 February 2025
Accepted: 10 February 2025
Published: 12 February 2025

Citation: Subahi, A.F. Enhancing Software Sustainability: Leveraging Large Language Models to Evaluate Security Requirements Fulfillment in Requirements Engineering. *Systems* **2025**, *13*, 114. https://doi.org/10.3390/systems13020114

Copyright: © 2025 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In today's digital era, cybersecurity serves as a vital pillar in preserving the fundamental components of our society. This role encompasses protecting national security, preserving digital privacy, safeguarding individuals, providing business continuity, and establishing digital trust. Cybersecurity not only protects our personal data and ensures the integrity of our digital infrastructure but also plays a critical role in the social sustainability of software systems [1].

1.1. Secure Software Design for Social Sustainability

The social sustainability of software focuses on maintaining and improving social quality and cohesion, and it is significantly influenced by cybersecurity. As our society becomes more digitally integrated, the reliability and security of digital software systems contribute to social trust and overall wellbeing. Inadequate cybersecurity and the absence

of software system security can lead to breaches that disrupt social services, data confidentiality, privacy, and integrity, thereby adversely affecting social sustainability [2,3]. Conversely, robust cybersecurity practices and efficient mechanisms for designing secure software systems can enhance social sustainability by fostering digital trust and inclusivity while mitigating risks. Thus, the intersection of cybersecurity and social sustainability represents a crucial area of focus in our increasingly digital society.

1.2. The Role of Non-Functional Requirements in Secure and Sustainable Software

From a software engineering perspective, non-functional requirements (NFRs) play a critical role in ensuring the development of secure, high-quality software systems that can significantly benefit society and its sustainability. Unlike functional requirements that define what a system should do, NFRs specify how the system should behave; they encompass aspects such as performance, security, usability, reliability, and maintainability, which are essential for the overall quality and user experience of sustainable software systems.

In the context of security requirements, NFRs dictate the measures that a system should have in place to protect against threats and vulnerabilities, including data encryption, user authentication, and access controls [4]. By identifying these requirements, we can build software systems that not only achieve their intended functionality but also ensure data protection, user privacy, and system robustness. These qualities are particularly crucial in sectors such as healthcare and other public services, where software systems can greatly affect social sustainability and wellbeing [4]. In essence, NFRs play a pivotal role in leveraging technology for societal benefit by driving the development of secure and high-quality software.

1.3. Research Motivation

While many LLM-based methods focus on identifying and classifying security vulnerabilities in software at the code level [5–11], there is a significant need for early security assessments in the software development lifecycle, especially during the requirements engineering (RE) phase. Our methodology fills this gap by systematically extracting and classifying security issues from NFR statements early on, providing a proactive security stance. Our approach categorizes security issues into seven groups, aligning with established security taxonomies to help prioritize and effectively target mitigation strategies.

It is strongly believed in the software engineering community that the RE phase is crucial, as it lays the foundation for all subsequent development activities. Overlooking security vulnerabilities at this stage can result in more severe consequences later, making detection and resolution more costly and complex. Our approach not only anticipates potential security issues early but also promotes a security-aware culture among stakeholders and developers from the start. Unlike traditional methods that respond to vulnerabilities after their introduction, our proactive methodology identifies issues early, allowing for more flexible and cost-effective changes.

In practice, the proposed approach can bring benefits to the development lifecycle in its early stages, guiding stakeholders and developers in making informed decisions about mitigation strategies, reducing the likelihood of costly and complex changes during implementation or maintenance.

1.4. Highlights of Research Contributions

This research contributes to the fields of software engineering and security, with a particular focus on the social sustainability of software systems; it introduces a proof-of-concept approach for assessing the fulfillment of security requirements during the requirements engineering phase. The approach highlights the significant role of large language models (LLMs) in detecting various security issues early in the development

process. By addressing these issues at an early stage, this approach aims to positively impact subsequent development phases and enhance the overall quality and security of the final system.

Although numerous studies have explored software requirements classification using various natural language processing (NLP) and machine learning (ML) techniques [5–9], we conducted an experiment to compare different models in order to identify the most effective one for detecting security-related issues by classifying NFRs. The proposed approach incorporates (1) a fine-tuned bidirectional encoder representation from a transformer (BERT) model, which is utilized and evaluated for categorizing software NFRs into two classes—security-related NFRs and non-security-related NFRs—and (2) sentence-BERT embeddings combined with a support vector machine (SVM) for a detailed security analysis strategy of security-related NFRs. The main contributions of this research can be summarized as follows:

1. The introduction of a proof-of-concept framework for assessing the satisfaction of security requirements during the requirements engineering phase, leveraging the capabilities of BERT-based LLMs;
2. The development and evaluation of a fine-tuned BERT model for classifying NFRs into security-related and non-security-related categories, ensuring the accurate identification of security requirements;
3. The proposal of a hybrid approach using sentence-BERT embeddings and an SVM for detailed security-related NFR analysis, providing a deeper understanding of security issues;
4. A comparison of multiple models and techniques to identify the most effective approach for detecting security-related issues, offering practical insights for software engineering and cybersecurity practices.

The rest of this paper is organized as follows: Section 2 presents related works, including state-of-the-art LLM prediction models, as well as related deep learning approaches and existing security requirements elicitation frameworks. Section 3 presents and discusses the proposed approach, including the definition of security requirements fulfillment (SRF) of software systems, the approach to measuring security levels, and risk-handling strategies. Section 4 demonstrates the materials and methods used, including the training dataset, as well as the preprocessing, labeling, and model development steps. Section 5 describes the two experiments conducted to evaluate the proposed approach, followed by results analysis, interpretation, and the findings of each experiment in Sections 6 and 7. Section 8 discusses the conclusions of this study and future research direction.

2. Related Works

2.1. State-of-the-Art LLM Prediction Models

In [10], several transformer-based language models for detecting software vulnerabilities in C/C++ code were assessed by using datasets such as VulDeePecker and SeVC, which include various types of vulnerabilities. Their findings revealed superior F1-scores for transformers, exceeding 95% in binary classification tasks on the VulDeePecker dataset and around 90% in multi-class tasks on the SeVC dataset, outperforming traditional models such as BiLSTM and BiGRU.

Additionally, in [11], the authors investigated the effectiveness of the pre-trained CodeBERT model versus a newly trained version for defect prediction across different software versions and projects. The evaluation, conducted on C/C++ vulnerability datasets such as VulDeePecker and SeVC, showed that the pre-trained CodeBERT significantly outperformed the newly trained model, achieving F1-scores above 95% in binary classification

and approximately 90% in multi-class detection tasks. This confirms the superiority of pre-trained transformer models such as CodeBERT in automated vulnerability detection.

Furthermore, the authors of [12] introduced V2W-BERT, a novel transformer-based framework aimed at automating the classification of software vulnerabilities into relevant categories in a hierarchical dictionary of weaknesses. Using data from MITRE and the National Vulnerability Database, the researchers employed techniques from natural language processing, link prediction, and transfer learning. V2W-BERT demonstrated superior performance, with up to 97% accuracy on randomly partitioned data and 94% on temporally partitioned data, significantly outperforming existing methods; it achieved an accuracy of 89–98% for common vulnerabilities and exposures, as well as 48–76% for rare ones, showing an improvement of 10–15% over previous approaches.

The authors of [13] introduced a BERT-CNN framework to classify the severity of software vulnerabilities, a critical aspect for prioritizing security interventions. Utilizing data from the National Vulnerability Database, this approach combines BERT to encode vulnerability descriptions and a CNN to highlight key features for severity classification. The BERT-CNN model achieved up to 97% accuracy in distinguishing between severe and non-severe vulnerabilities, surpassing previous methods based on traditional ML or simpler neural architectures. This effectiveness is particularly valuable in prioritizing severe vulnerabilities in security efforts.

Overall, the primary focus of the state-of-the-art works discussed in [10–13] is to demonstrate the power of pre-trained LLMs in automating vulnerability detection during or after the coding phase of software development. This leaves a critical gap in addressing security vulnerabilities earlier in the software lifecycle, particularly during the requirements engineering (RE) phase.

2.2. Related Deep Learning Approaches

Two integrated models were utilized in [14]. First, deep neural networks (DNNs) were used to predict the causes and sizes of wildfires, achieving accuracy rates of 76.9% and 76.4% for five-class classification tasks, respectively. Second, a forecasting model based on long short-term memory (LSTM) networks and autoencoders was applied to predict the likelihood of imminent wildfires in high-risk regions such as California. This model demonstrated robust performance, with a coefficient of determination of 0.90 and a root-mean-square error of 49.5076 for one-week forecasts.

The authors of [15] introduced a novel framework that combines deep reinforcement learning with Monte Carlo tree search to solve both single-objective sequential decision problems and multi-objective sequential decision problems. This framework efficiently navigates complex decision spaces, while the deep neural network component learns optimal policies through experience. This integration enables the system to balance conflicting objectives, such as minimizing maintenance costs while ensuring operational flexibility and safety.

In summary, the approaches discussed in [14,15] emphasize the importance of integrating predictive models early to mitigate risks and optimize outcomes. Similarly, the proposed research applies this proactive mindset to software security by leveraging deep learning to identify security vulnerabilities during the RE phase. Moreover, in the long term, it optimizes the flexible operations to mitigate and handle possible risks in their relevant domains.

2.3. Existing Security Requirements Elicitation Frameworks

Eliciting the appropriate security requirements at the early stages of software development, or in the RE phase, is important for identifying the needs and more advanced

security concerns of modern software systems. The security threat-oriented requirements engineering (STORE) methodology [16] is a security requirements elicitation technique employed in the RE phase, based on the analysis of security threats; its goal is to elicit more efficient security requirements using four identified key points for analyzing attacks: point of attack, point of belief, point of conjecture, and point of dependency. The STORE methodology has been compared with other techniques, such as system quality requirements engineering (SQUARE) and the model-oriented framework for security requirement engineering (MOSRE), and has demonstrated its superior effectiveness and efficiency in eliciting security requirements.

The TrUStAPIS technique, based on JavaScript Object Notation (JSON), is another requirements elicitation method [17] that seeks to enhance the trust level of Internet-of-Things (IoT) systems by assisting developers in eliciting effective security needs throughout the early stages of the IoT entity development lifecycle. By employing a JSON template, developers have the ability to determine which requirements need to be retained during the elicitation process. TrUStAPIS ensures trust by assessing six essential domains of NFRs: security, usability, privacy, availability, safety, and identity. It is worth noting that the methods presented in [16,17] lack the automation and scalability offered by LLM approaches.

3. Proposed Approach

Designing an approach that integrates LLMs with machine/deep learning into the RE phase to measure the security level of software is a strategic initiative to proactively identify and address the potential security risks that might be detected during this phase, thus improving the overall security attitude of the developed software, unlike traditional approaches that focus on post-development vulnerability detection or manual elicitation techniques at later stages of the development lifecycle. This can bridge research gaps identified across all areas covered in the Related Works Section.

3.1. Definition of Security Requirements Fulfillment (SRF) of Software Systems

The main categories of security requirements to identify the major classes of SRF considered in the proposed framework are presented in this section. A feature model [18] is used to represent the basic categories and security terminologies, or requirements, using the notion of a hierarchical feature diagram structure. In the context of software engineering, a feature model can be defined as a structured representation of all of the products in a software product line, along with their features. These models are used in domain analysis and product line engineering to identify and illustrate commonalities and variabilities within a product line. The features are typically organized in a hierarchical manner, with the relationships among them denoted as mandatory, optional, or alternative [18]. Figure 1 demonstrates the main classes of security issues developed in the proposed framework that may appear and be detected in the software requirements analysis phase (described in Table 1).

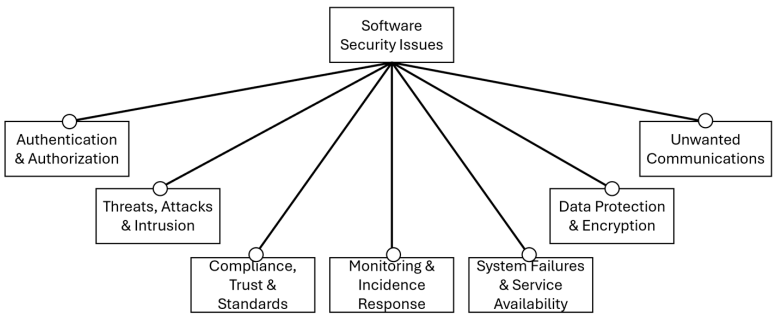


Figure 1. Top-level feature diagram demonstrating the main software security issues.

Table 1. Security issues developed in the proposed framework.

Security Issue	Definition
Authentication and Authorization	Valid users must be authenticated to access the system or its parts; however, "authorization" grants or denies access rights to authenticated users and controls what those users can do in the system [19].
Data Protection and Encryption	Data must be secure, available, and accessible to authorized users, retaining their integrity and confidentiality [20].
Compliance, Trust, and Standards	The system follows laws and standards. Trust refers to users' confidence to work well and manage their data appropriately. User adoption and system utilization depend on trust. In addition, the system follows global technological specifications or procedures that guarantee interoperability, compatibility, and reliability [21].
Monitoring and Incident Response	Monitoring means continual observation. Common processes include identifying and analyzing incidents, containing and eliminating threats, recovering from incidents, and carrying out post-incident assessments to learn from events and to improve future responses [22].
Threats, Attacks, and Intrusion	Threats can generate unwanted incidents that harm systems and exploit system weaknesses, compromising security. Hackers trying to break into systems are a threat. An attack is an action that exploits a vulnerability to compromise a system's security; it is an attempt to disrupt system operations, steal data, or disable the system. An intrusion is unauthorized system access after a successful attack [23].
System Failures and Service Availability	Hardware or software failures, network issues, and cyberattacks may cause major system failures, which can disrupt services and result in data loss. In contrast, the capacity of a system or service to be available and operational when needed is called "service availability" [24].
Unwanted Communications	Any disruptive or unnecessary software system messages, alerts, or interactions are referred to as "unwanted communications". Allowing users to customize notification settings, preventing spam, and making system communications relevant and required may assist with this [25].

3.2. Measuring the Security Level of Software Systems

In this section, a four-step approach for measuring the security level of software systems in the RE phase is developed and discussed. The Unified Modeling Language (UML) activity diagram in Figure 2 illustrates the main steps of the proposed framework. The following list presents a step-by-step guide on how to apply the proposed framework:

1. Software requirements collection: Gather all written software requirements from the software requirements specification document belonging to the system of interest. This should ideally cover all of the system's functionalities and characteristics;
2. NFR classification: this step includes identifying the security requirements that are essential for the system, extracting them, and determining their associated types;
3. Security-related NFR analysis: In this step, the extracted list of security-related NFRs is evaluated to diagnose some security vulnerabilities;
4. Risk management: After analyzing the security requirements, identifying their categories, and understanding the security needs of the system, possible related risks or vulnerabilities are identified. Next, for each identified risk, suggested mitigation strategies are provided. These strategies may involve some modifications of the requirements statements, or even adding certain requirements if necessary;
5. Security-related NFR improvement: Improve the initial requirements list to better cover the identified risks based on the risk assessment and mitigation strategies. This could mean adding new requirements, modifying existing ones, or removing those that introduce unacceptable risks.

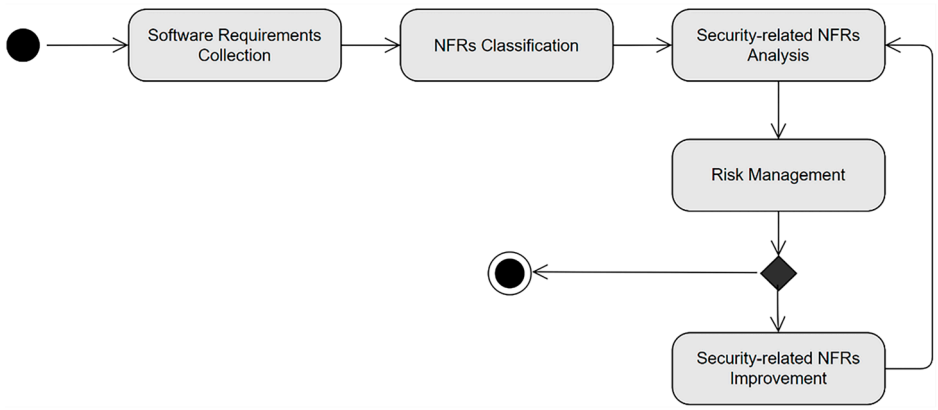


Figure 2. Main steps of the proposed security level measurement framework.

3.3. Strategies for Handling Risks

In the proposed framework, two steps are defined for handling risks: the identification of possible risks, and the recommendation of risk mitigation strategies. Risk mitigation strategies are crucial components of any software system's security framework aimed at reducing the impact of threats to manageable levels. These strategies are vital because they help organizations proactively address uncertainties that could compromise the reliability, accessibility, or confidentiality of their software systems. As depicted in Figure 3, our framework outlines general risk management recommendations based on each class of SRF.

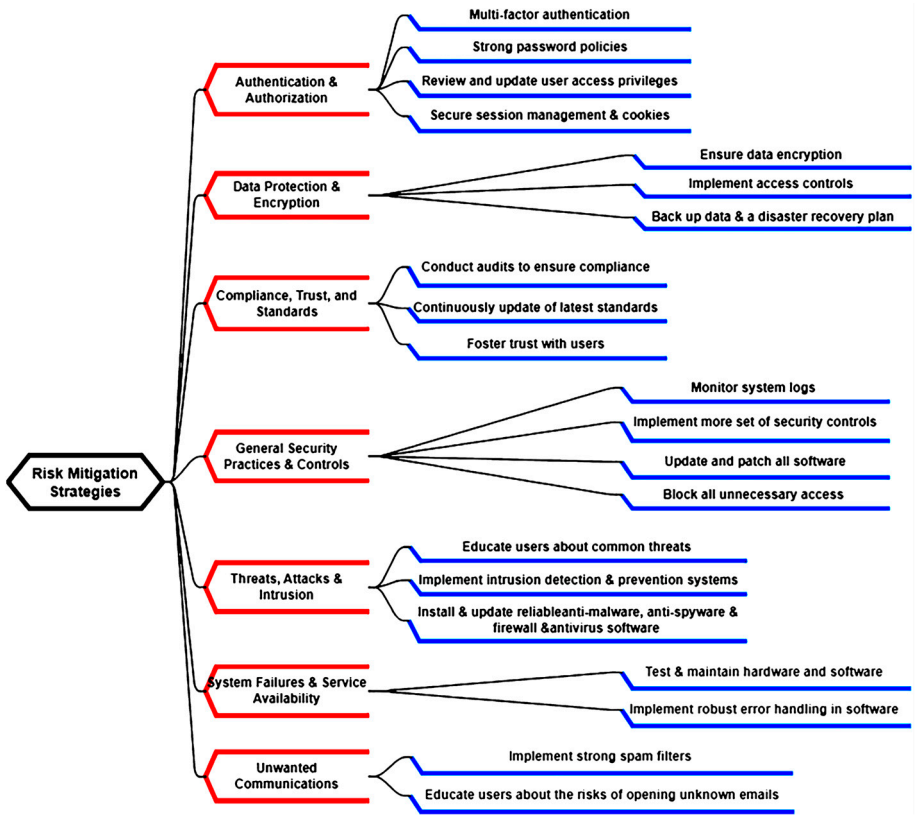


Figure 3. Strategies for risk management.

Given that our framework is designed for implementation during the requirements engineering phase, these strategies are particularly effective in enhancing system requirements. For example, if vulnerabilities related to data integrity or confidentiality are detected (possibly due to inadequate coverage by existing NFRs), the list of NFRs may need to be revised. New NFRs that specifically address the handling of sensitive information, guided

with respect to the related mitigation strategies, can be added. This proactive measure not only rectifies immediate security gaps but also sets a secure foundation for subsequent development activities, ensuring a robustly secure system from the outset.

4. Materials and Methods

This section highlights the data resources utilized and the core steps undertaken for applying the proposed methodology in this research, including the training dataset used, the preprocessing steps, and labeling techniques. Furthermore, the major stages for model development are also discussed in this section, as illustrated in Figure 4.

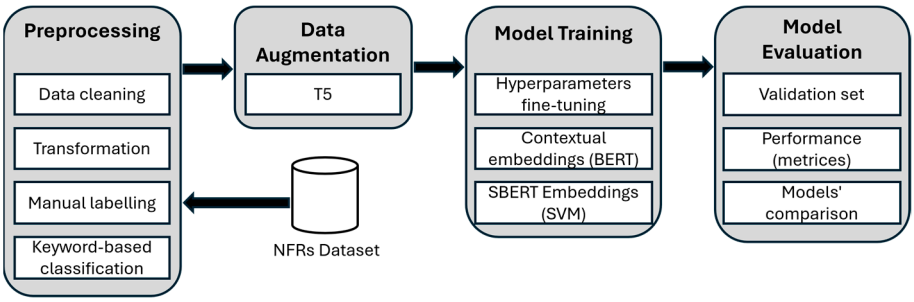


Figure 4. Proposed model architecture.

4.1. Training Dataset and Preprocessing

We considered the extended version of a non-functional requirements dataset based on the commonly used PROMISE [26] and PROMISE_exp [27,28] as the basis for this research. The original version of the PROMISE dataset contains 625 requirements, whereas PROMISE_exp has 969 requirement statements (labeled) written in the English language. Both repositories have been used in different studies [28–30] to address the problem of software requirements classification, including security requirements.

The extended version was developed in a previous work [28], mainly to improve the process of evaluating software sustainability by incorporating additional functional requirements and NFRs, focusing more on specific NFR types, such as reliability (RE), efficiency (EF), accessibility (AC), and interoperability (IN), which were not fully represented in the original datasets (PROMISE and PROMISE_exp). The expansion was carried out in two stages, focusing on NFR types. The data collection process involved retrieving publicly available documents, including student projects and dissertations related to software engineering and system development, via online searches.

Relevant documents were filtered, and requirement sentences were extracted from the requirement and analysis sections. These sentences were manually categorized as either FRs or specific types of NFRs. To ensure quality, all extracted requirements underwent validation for linguistic clarity and correctness by a software engineer or computer scientist before being added to the dataset. The preprocessing steps included cleaning, transforming, and labeling the data to ensure their quality and representativeness. A manual labeling technique was employed, with a domain expert assigning labels that align with software sustainability contexts and green IT practices.

It should be noted that the structure of both repositories has three main attributes: ProjectID, RequirementText, and Class. Each ProjectID attribute expresses the unique number that refers to the project from which the requirement statement was taken; the RequirementText attribute denotes the real text utilized to represent a software system requirement in a specific project document, and the Class attribute represents the classified type for each RequirementText. In addition, the dataset developed in another study [28] contains an additional labeling column for software sustainability classification. Two

informative labels were used for expressing two green information technology factors: eco-technical and socioeconomic.

4.2. Dataset Labeling Technique

Two meaningful labels were introduced to express the seven classes of SRF, inspired by the major software security concerns previously identified, categorized, and discussed above. These labels were used for semantic text classification, which classifies semantic, NFR statements into one of two categories (security NFR or non-security NFR). Additionally, in this work, we followed the keyword-based text classification approach [31], which involves the utilization of appropriate NLP techniques and a predefined list of critical terms to classify the data in a dataset. The UML activity diagram displayed in Figure 5 demonstrates the overall labeling process.

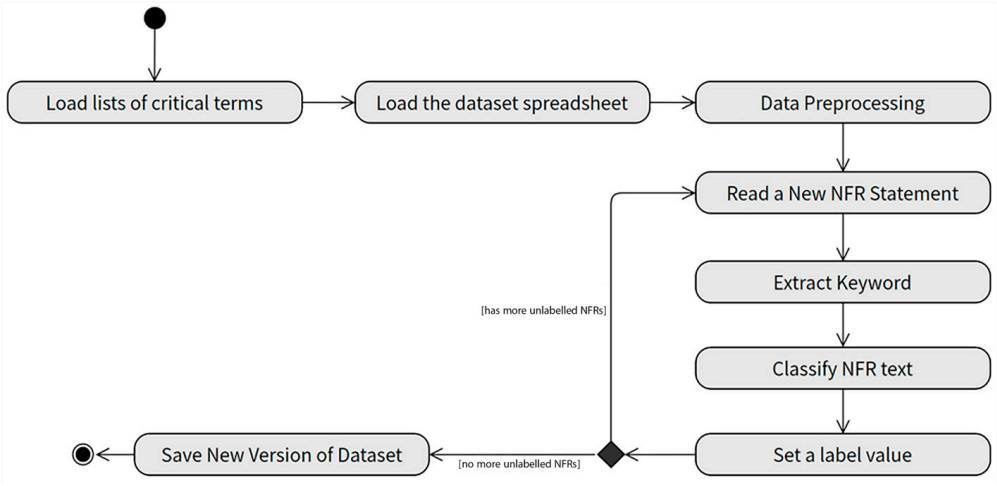


Figure 5. Process of the dataset labeling technique.

After completing the labeling process, all labeled security-related NFRs were validated and checked for any exceptional mislabeling. A manual review was conducted to examine each labeled NFR statement in order to ensure alignment with the defined category. Out of the 803 statements, only ~2.4% required manual relabeling due to unclear phrasing that did not fit the predefined categories. All validated requirements were then saved into the dataset. This new version of the dataset consisted of 803 (original and not augmented) NFR requirements distributed over the two classes, with 367 of them being security NFR statements and 427 being non-security NFR statements, as shown in Figure 6.

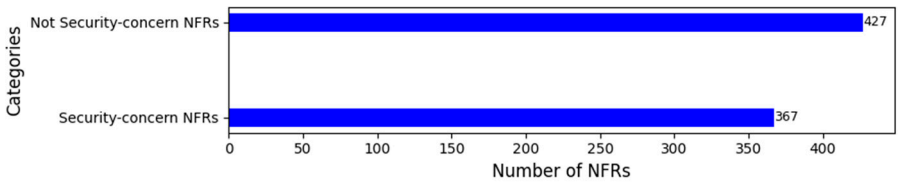


Figure 6. Category distribution of NFR statements in the new version of the dataset.

4.3. Model Selection and Development

One of the main aims of this work was to investigate the role of large language models in the classification of software security requirements. Four types of LLM models were considered for use in the NFR classification stage of the proposed approach: DistilBERT, original BERT, LTSM, and CNN, which were developed to guide the first research experiment. The UML activity diagram in Figure 7 illustrates the NLP pipeline of the BERT-based model developed for the NFR classification phase of the proposed approach.

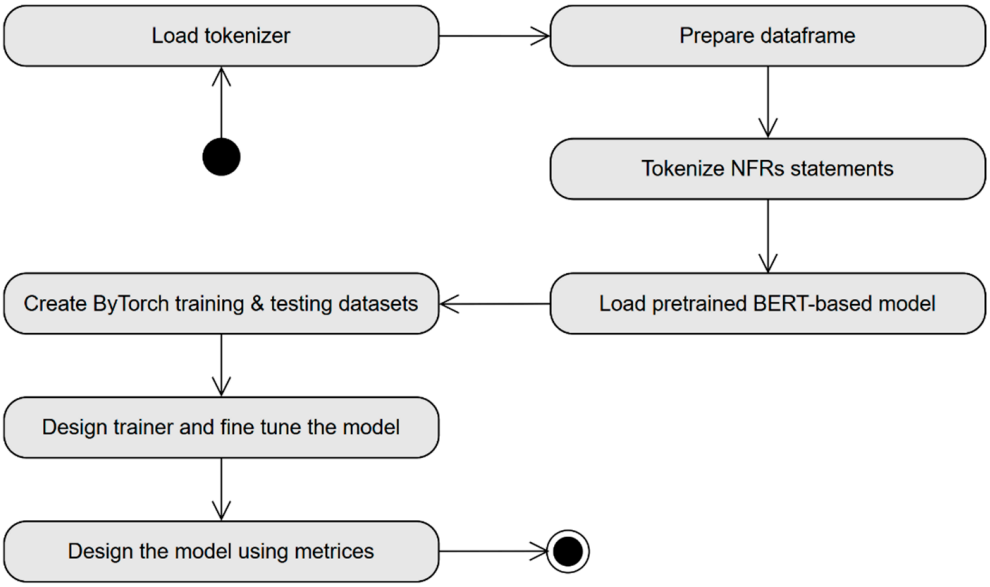


Figure 7. BERT-based model development and evaluation process.

4.3.1. LLM Model Selection

It should be noted that the choice of the BERT and DistilBERT LLMs for the targeted classification task was based on their established ability to capture contextual information in a bidirectional manner. BERT's transformer-based architecture has markedly enhanced various NLP tasks, attaining state-of-the-art performance through the utilization of bidirectional attention mechanisms, as discussed in [32]. This enables BERT to recognize the links between words in a sentence more efficiently than classic unidirectional models, making it especially appropriate for tasks such as text classification.

Furthermore, DistilBERT was selected for its efficiency, with minimal fine-tuning, reduced size (40%), and a faster performance (60%), while retaining the language understanding capabilities (97%) of BERT and other LLMs [33]. In [33], it is highlighted that DistilBERT provides a smaller and quicker alternative to BERT, making it suitable for resource-limited settings without substantially sacrificing accuracy.

It should be noted here that, despite the superior performance of the widely used deep neural networks when compared with many classic deep learning models based on statistics, they represent texts using word vectors that have been pre-trained on a large corpus and are typically employed as sequence features. Typically, these vectors are trained using the word2vec tools, which assume that words with similar meanings tend to appear in similar contexts [34]. In contrast, as the transfer learning capabilities of large-scale pre-trained models in BERT-based LLMs become more dominant in the NLP domain, classical pipelines in NLP-based text classification systems can be replaced [34].

Furthermore, DistilBERT improves computational efficiency and parallelism by utilizing its transformer-based architecture to handle long-range text dependencies. The benefits of this approach exceed those of alternative methods adopted in typical NLP, such as bag-of-words, term frequency–inverse document frequency, word embedding, recurrent neural networks, and long short-term memory [35,36].

In addition to the selected BERT-based models, two classic deep learning models were considered: LTSM and CNN. According to [37,38], LSTM models are designed with the ability to capture long-term dependencies and sequential patterns, which are essential for understanding contextual relationships in text, such as the flow and meaning of sentences in toxic comment classification. Their ability to address the vanishing gradient problem ensures effective learning from longer text sequences. CNNs, on the other hand, are highly

efficient at extracting local features and patterns, such as n-grams or key phrases, through their convolutional filters. As highlighted in [37,38], CNNs' computational efficiency and their ability to process text at a granular level make them especially useful for identifying critical features in classification tasks.

4.3.2. ML Model Selection

In addition, various ML models (e.g., LR, SVM, and XGBoost) were also utilized in the proposed experiments. For instance, the work presented in [39] highlights that XGBoost's gradient-boosting framework excels at capturing intricate patterns in data through its ensemble of decision trees, making it particularly effective for distinguishing between multiple attack types. Furthermore, its scalability and capacity to optimize performance with limited computational resources make it an ideal choice for intrusion detection systems, where both accuracy and efficiency are paramount. These qualities ensure that XGBoost can address the challenges posed by complex, imbalanced datasets while maintaining high performance in critical security applications.

According to [40], LR is particularly well suited for various text classification tasks; it provides probabilistic outputs and can efficiently handle linear relationships between features, making it a reliable choice for straightforward classification tasks. The SVM, on the other hand, excels in separating classes with a clear margin by utilizing kernel functions, making it effective for both linear and nonlinear classification problems. The authors emphasized that both models perform well in multi-label text classification, especially when dealing with smaller datasets or less complex tasks.

4.3.3. Fine-Tuning of Selected Models

The fine-tuning strategy for the utilized BERT-based models is based on selecting key hyperparameters, such as the learning rate, warm-up steps, weight decay, batch size, and number of epochs, to create a balance between computational efficiency and model performance. The learning rate of approximately (3.26×10^{-5}) was chosen using grid search techniques, as suggested in [41], to achieve a balance between convergence speed and avoid overfitting. Moreover, warm-up steps (500) and weight decay (0.01) were used to further enhance the generalization and convergence of the models [42]. Additionally, a small batch size of 8 was used, which has been shown to stabilize training with limited computational resources while still enabling effective gradient updates. The model was fine-tuned for 8 epochs, allowing for sufficient exposure to the training data without overfitting [43].

Aside from that, the fine-tuning step for the LSTM model focused on different hyperparameters, such as hidden layer sizes and embedding, dropout, gradient clipping, and optimizers. Similarly to those in [44], the embedding and hidden layer sizes were set to 128 in order to establish a balance between efficiency and the model's ability to capture semantic patterns. The dropout was set to 0.2 to reduce overfitting. Gradient clipping (0.1) and the Adam optimizer (learning rate 0.001) were utilized to ensure stable training and faster convergence of the model [45,46].

Regarding the CNN model, the embedding dimensions and convolutional layers were selected with 128 filters to extract text features efficiently. Contextual information was captured using two convolutional layers with a kernel size of 3, while the adaptive max-pooling was set to ensure the consistency of feature map sizes. Furthermore, the CNN model's training included the Adam optimizer, with a learning rate of (3.26×10^{-5}), a gradient clipping of (0.1), and a weight decay of (0.01), to prevent overfitting and enhance generalization [42].

4.3.4. Data Augmentation

To enhance the performance of the model on our limited dataset in the SRF analysis stage of the proposed approach, we employed data augmentation through paraphrasing using the T5-small model [47–49]. T5-small, a transformer-based text-to-text pre-trained model [48], was utilized to generate semantically equivalent paraphrases of the original security NFR statements. This process enriched the dataset by introducing linguistic diversity while preserving the core meaning of the statements, enabling the model to generalize better across variations in phrasing. The paraphrased data were combined with the original dataset during training, reducing overfitting and improving the model's classification accuracy for NFRs. The UML activity diagram in Figure 8 illustrates the NLP pipeline of the BERT-based model used in the SRF analysis phase of the proposed approach.

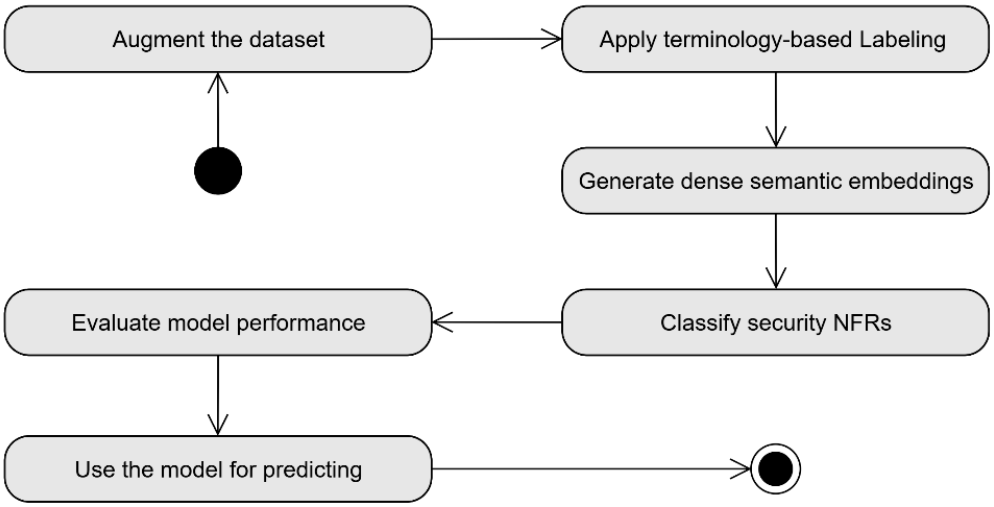


Figure 8. Summary of the ML pipeline process utilized in the SRF analysis stage of the approach.

Moreover, knowledge-based labeling, as a method that leverages predefined semantic clues to guide classification models, has been recognized for its ability to improve the reliability of multi-class classification tasks [50–52]. This mechanism is adopted in the proposed approach, in which it can play a crucial role in improving multi-class classification by using a predefined list of critical terms associated with specific NFR categories (Table 2). This increases the model's ability to accurately identify and label requirements, as the predefined terms serve as semantic cues that guide the model in associating statements with the correct NFR category, even in cases of linguistic variability or ambiguous phrasing.

Table 2. Knowledge-based labeling using predefined critical terms.

Security Issue	Related Terms
Authentication and Authorization	Authentication, biometrics, password, role, permission, authorization, session, cookies, and privileges.
Data Protection and Encryption	Protection, hash, verify, encryption, secure, sign, privacy, sensitive, backup, recovery, and cryptography.
Compliance, Trust, and Standards	Compliance, guidelines, responsibilities, regulations, comply, support standards, audit, trustworthy, security control, trust, policy, safe, and security.
Monitoring and Incident Response	Measure, logout, monitor, track, mitigate risk, alert, preventing access, control access, and block access.
Threats, Attacks, and Intrusion	Malicious, vulnerabilities, leakage, injection, attack, hack, spy, intrusion, denial of service, phishing, spoofing, Trojan, worms, threat, fraud, fake, malware, spyware, and hijacking.
System Failures and Service Availability	Fail, available, fault, error, tolerance, failback, failure, failover, disruption, crash, unavailability, update, and damage.
Unwanted Communications	Unsolicited, undesirable, unwanted, and spam.

Additionally, we utilized pre-trained sentence-BERT (SBERT) embeddings in conjunction with an SVM, a lightweight ML classifier. SBERT embeddings effectively mitigate the challenges of small datasets by leveraging the pre-trained BERT model to generate dense vector representations of sentences [53,54]. These embeddings capture rich semantic information, enabling robust generalization even with limited labeled data, while reducing the risk of overfitting. By fine-tuning on task-specific data, SBERT enhances classification performance without requiring extensive labeled datasets, as its pre-trained contextual knowledge compensates for data scarcity [53,54].

4.4. Model Implementation

The proposed approach was implemented using Python (3.11.11) in the Google Colab environment. We employed Pandas and NumPy for data manipulation and preprocessing. The scikit-learn library was utilized for dataset partitioning and performance assessment metrics such as accuracy and classification reports.

We utilized the SentenceTransformers library to produce high-quality semantic representations of the input text for managing text embeddings. We employed XGBoost for the machine learning model because of its efficiency and performance in classification tasks. Furthermore, for experiments involving sequence-to-sequence language models, we employed the Transformers library from Hugging Face. The experiments utilized Python's built-in random module (import random) to control for randomization.

5. Evaluation

The protocol for evaluating the text classification performance of the selected BERT-based LLMs and the ANN models is discussed in this section. We adopted a fine-tuning strategy to train these models on the dataset, which contained examples of NFR phrases classified into two classes: security NFR and non-security NFR statements. It should be noted that the dataset utilized in the designed experiment comprised 803 (original and not augmented) samples of NFR textual statements, of which 427 were labeled "non security NFR" and 367 were labeled "security NFR".

5.1. Experiment (1) Objective

Given the stated hypothesis, the primary objective of the experiment designed for this study was to assess the ability of BERT-based LLMs to classify software NFRs into one of two predefined classes, in comparison to ANNs. This objective was achieved through the following tests: (1) assessing the models' ability to achieve the highest possible performance scores across the metrics of accuracy, precision, recall, and F1-score, whereby these performance metrics were quantified to precisely evaluate the models' effectiveness in making accurate predictions on the validation set, and (2) assessing the model's performance through ROC-AUC scores to enable a comprehensive evaluation of their ability to distinguish between classes effectively. This metric was quantified to accurately determine the models' proficiency in ranking predictions correctly across all possible classification thresholds on the validation set.

Similar to the evaluation protocol that was previously applied and discussed in another study [28], the data were divided into training (70%) and testing (30%) datasets in this experiment. A hyperparameter tuning plan was measured to enhance the model's performance using two core parameters: batch size and number of epochs. "Batch size" denotes the number of training instances utilized in a solo update in the training procedure, which affects the performance. It is worth mentioning that the works discussed in [55,56] indicated that smaller batch sizes (eight and sixteen) often produce better test accuracy and generalization performance compared to larger batch sizes. The investigation in [56]

found that as the batch size increased from 16 to 128, the average combined false positives plus false negatives increased by 15.4%, and the average model F1-score decreased by 1%. Thus, in this experiment, the model was trained over two batch sizes (eight and sixteen) and a number of epochs ranging from two to nine. Since selecting a smaller number of epochs can be beneficial, training a model for too many epochs can lead to overfitting, where the model becomes overly specialized to the training data and performs poorly on unseen data. A smaller number of epochs can prevent this issue and ensure that the model generalizes well to new data. Furthermore, using a smaller number of epochs can reduce computational costs and save training time [57].

5.2. Experiment (2) Objective

The objective of this experiment was to evaluate the effectiveness of machine learning models in analyzing security NFRs and classifying them into the seven predefined categories. The focus was on comparing the performance of three models—logistic regression (LR), an SVM, and XGBoost—to determine their ability to handle the diverse and nuanced nature of security NFRs. The models were assessed using standard evaluation metrics, including precision, recall, F1-score, and accuracy, to capture their ability to make accurate predictions, minimize false positives, and identify all relevant instances. The goal was to identify the model that best balances these metrics across all categories, providing a reliable and efficient method for automating the classification of security NFRs.

6. Results, Analysis, Interpretation, and Findings of Experiment (1)

The analysis aimed to determine which model performs best for the task at hand, based on these metrics. Thus, in this section, we assess and compare the performance of the four selected machine learning models: DistilBERT, BERT, LSTM, and CNN. Each model was evaluated over two batch sizes and a range of epochs using six key performance metrics: accuracy, precision, recall, F1-score, training loss, and ROC-AUC. We initially examined two batch sizes (eight and sixteen), alongside various hyperparameters such as learning rate and weight decay. The Bayesian hyperparameter optimization framework (Optuna) was utilized to tune the learning rate. It was found that the overall results with a batch size of eight were superior to those with a batch size of sixteen.

After tuning the hyperparameters, we continued our experiment using a batch size of eight. We ran the experiment across various epoch numbers within the range. Tables 3–5 display the performance scores obtained after conducting the experiment.

Table 3. Performance metrics of the BERT model with different numbers of epochs and a batch size of 8.

Epoch	2	3	4	5	6	7	8	9
Accuracy	0.7391	0.8261	0.8571	0.8447	0.8447	0.8820	0.8198	0.8075
Precision	0.7399	0.8402	0.8570	0.8494	0.8518	0.8828	0.8372	0.8193
Recall	0.7391	0.8261	0.8571	0.8447	0.8447	0.8820	0.8198	0.8075
F1-Score	0.7394	0.8219	0.8571	0.8430	0.8425	0.8815	0.8164	0.8038

Table 4. Performance metrics of the DistilBERT model with different numbers of epochs and a batch size of 8.

Epoch	2	3	4	5	6	7	8	9
Accuracy	0.6894	0.7888	0.7826	0.8820	0.7764	0.8137	0.7640	0.7763
Precision	0.7003	0.7929	0.8028	0.8818	0.7838	0.8206	0.7887	0.7815
Recall	0.6894	0.7888	0.7826	0.8820	0.7764	0.8137	0.7640	0.7764
F1-Score	0.6896	0.7889	0.7748	0.8819	0.7720	0.8106	0.7536	0.7728

Table 5. Performance metrics of the LTSM model with different numbers of epochs and a batch size of 8.

Epoch	2	3	4	5	6	7	8	9
Accuracy	0.7081	0.6832	0.7640	0.7764	0.7143	0.7205	0.7329	0.7205
Precision	0.7024	0.6630	0.7416	0.8261	0.7465	0.7308	0.7209	0.7195
Recall	0.7284	0.7531	0.8148	0.7037	0.6543	0.7037	0.7654	0.7284
F1-Score	0.7152	0.7052	0.7765	0.76	0.6973	0.7170	0.7425	0.7239

6.1. Performance Metrics Analysis

We attempted to clearly visualize the models' evolution in terms of accuracy, precision, recall, and F1-score.

According to the accuracy scores demonstrated in Figure 9, DistilBERT reached its highest accuracy of 88.20% at epoch 5; however, subsequent epochs revealed fluctuations in its performance, indicating potential instability or overfitting issues. In contrast, BERT demonstrated greater consistency in accuracy, achieving a peak of 88.20% at epoch 7, which suggests better generalization across different datasets. On the other hand, LSTM exhibited a significantly lower peak accuracy of 77.64%, reflecting its relative inefficiency for this task compared to the transformer models. The CNN model, achieving a maximum accuracy of 80.50% at epoch 5, also underperformed relative to BERT and DistilBERT, underscoring its limitations in capturing complex patterns as effectively as the more sophisticated architectures.

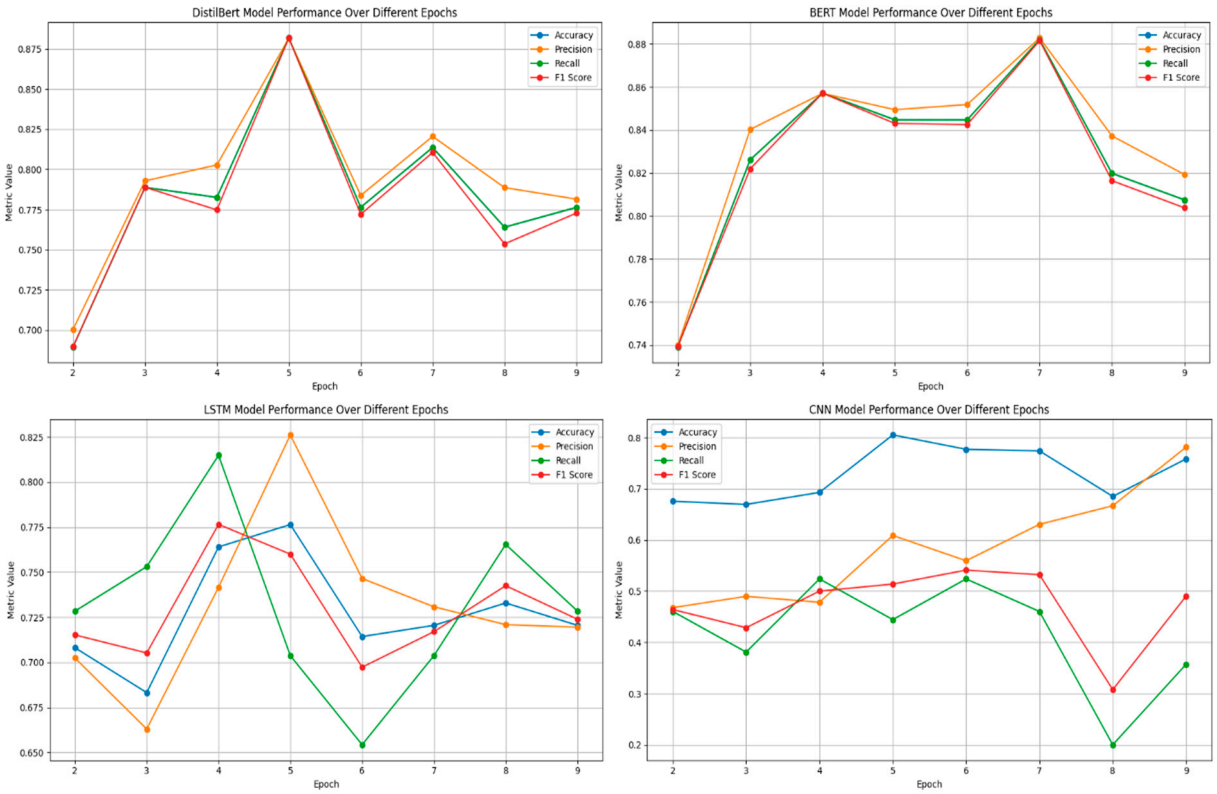


Figure 9. Performance comparison of BERT, DistilBERT, LSTM, and CNN across accuracy, precision, recall, and F1-score.

Moreover, precision metrics provide insights into the models' ability to minimize false positives. Both DistilBERT and BERT showed high precision, with peaks at 88.18% and 88.28%, respectively, indicating their effectiveness in identifying relevant instances with high confidence. However, LSTM and CNN recorded lower precision rates, peaking at

82.61% and 78.12%, respectively. These figures suggest that while LSTM and CNN can be useful in certain contexts, they may not be as reliable for tasks requiring high precision due to their greater propensity for false positive classifications.

In addition, recall is critical for assessing the models' capability to identify all relevant instances. BERT showcased very consistent recall rates, closely aligning with its accuracy figures, which highlights its ability to balance identifying relevant instances without introducing many false negatives. DistilBERT's recall closely mirrored its accuracy, peaking at 88.20%, suggesting effective sensitivity in detecting positive samples. In contrast, LSTM and CNN exhibited lower and more volatile recall rates, which could be problematic in applications where missing important instances carries significant consequences.

Furthermore, the F1-score, which balances precision and recall, is particularly informative for evaluating the overall performance of classification models. BERT maintained a robust F1-score, peaking at 88.15%, indicating its strong performance across both the precision and recall dimensions. DistilBERT also achieved a high F1-score of 88.19% at epoch 5, reinforcing its capabilities despite some noted instability in accuracy. However, LSTM and CNN lagged significantly behind, with their best F1-scores at 77.65% and 54.10%, respectively. These lower scores reflect the challenges that these models face when simultaneously optimizing for precision and recall, particularly in complex classification tasks.

The greater performance of BERT, as noted in the results, can be attributed to its ability to capture semantic details in the data through pre-trained contextual embeddings. This significantly influenced its precision, recall, and F1-scores. BERT achieved high scores in F1 and precision, demonstrating its strong ability to balance identifying relevant instances (recall) while minimizing false positives (precision). Additionally, LSTM and CNN recorded much lower F1-scores, reflecting their difficulty when simultaneously optimizing for both precision and recall.

In addition, BERT showed its ability to generalize across categories despite the dataset's imbalanced distribution of security-related NFRs. This shows its ability to handle this imbalance better, achieving consistently high accuracy and recall, unlike LSTM and CNN, which showed lower and more unstable recall and precision scores. These results highlight how BERT's pre-trained embeddings and self-attention mechanisms outperform LSTM and CNN in managing data complexity and imbalance.

6.2. Training Loss Analysis

The training loss metrics for BERT and DistilBERT demonstrate the effective learning capabilities of these models over successive epochs. BERT, in particular, exhibits a consistent decline in training loss, reaching minimal levels in the later epochs, as shown in Table 6. This trend not only indicates successful learning from the training data but also highlights BERT's superior ability to mitigate the risk of overfitting. The model's robust architecture seems to effectively generalize across different datasets without memorizing noise, thereby maintaining high performance in unseen data scenarios.

Table 6. Training loss comparison between the BERT and DistilBERT models.

Model/Epoch	2	3	4	5	6	7	8	9
BERT	0.5532	0.3886	0.1927	0.2461	0.1062	0.1374	0.0013	0.0013
DistilBERT	0.6027	0.4246	0.4246	0.2399	0.0767	0.0851	0.0024	0.0011

In contrast, while DistilBERT also shows a significant reduction in training loss, it does not match the minimal loss levels achieved by BERT in later epochs. This suggests that while DistilBERT effectively learns from the training data, it may not be as efficient as BERT in preventing overfitting, which could be attributed to its simplified model structure,

designed for faster computations but at the potential cost of some learning depth. This is illustrated in Figure 10.

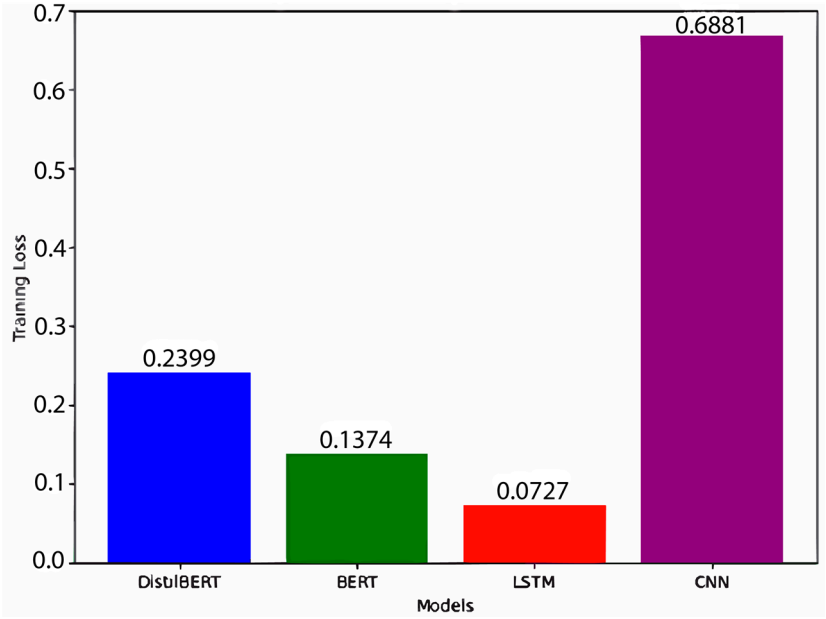


Figure 10. Training loss comparison across the models.

The LSTM and CNN models present a different scenario in their training loss trends. Both models start with higher initial losses compared to BERT and DistilBERT, indicating potential challenges in adapting to the dataset or inefficiencies in model architecture for the task at hand. Moreover, the rate of loss reduction for these models is slower, which points to less effective learning and optimization processes throughout the training phase. LSTM, while capable of capturing time-dependent information, may struggle with the complexity and size of the data, leading to slower and less effective convergence, as shown in Table 7.

Table 7. Training loss comparison between the CNN and LSTM models.

Model/Epoch	2	3	4	5	6	7	8	9
CNN	0.6792	0.6750	0.6707	0.6681	0.6663	0.6631	0.6603	0.6598
LSTM	0.4714	0.2988	0.1309	0.0727	0.0327	0.0125	0.0153	0.0003

Similarly, CNN, primarily known for its performance in spatial data interpretation, might encounter limitations when processing the sequential or contextual nuances of the dataset, resulting in less satisfactory training loss reductions. These observations underscore the need for more sophisticated or tailored model architectures, such as BERT, for tasks demanding high levels of abstraction and generalization, as demonstrated in Figure 10 above.

The consistently lower training loss demonstrated by BERT reflects its strong learning capability and its ability to generalize across categories without overfitting. This is supported by its better performance in recall and precision. In contrast, the LSTM and CNN models show higher initial training losses and slower convergence rates, leading to lower results across multiple metrics.

The lower recall value for LSTM reflects its difficulty in identifying all relevant instances, while its precision score indicates a moderate ability to minimize false positives. Similarly, CNN shows significant challenges, with low recall and precision scores, resulting in poor overall performance. These weaknesses are likely due to the inability of LSTM and

CNN to effectively handle the sequential dependencies and semantic complexity of the dataset. The imbalanced distribution of security-related and non-security-related NFRs further amplifies these challenges.

6.3. ROC-AUC Score Analysis

BERT stands out with the highest score among the evaluated models, achieving an ROC-AUC of 92.45% (Table 8). This score is indicative of BERT's superior discriminative ability, reflecting its effectiveness in distinguishing between classes. This high score suggests that BERT not only accurately identifies the positive class but also maintains a low false positive rate across various thresholds. This characteristic is crucial for applications where the cost of false positives is high, further underscoring BERT's applicability in sensitive or critical decision-making environments.

Table 8. ROC-AUC score comparison of the BERT and DistilBERT models.

Model/Epoch	2	3	4	5	6	7	8	9
BERT	0.8166	0.9004	0.9191	0.8847	0.8822	0.9245	0.9069	0.8747
DistilBERT	0.7232	0.8642	0.8366	0.9212	0.8822	0.8492	0.8496	0.8312

DistilBERT also demonstrates commendable performance, with an ROC-AUC of 92.12%. This score confirms that, despite its reduced complexity, DistilBERT retains much of the discriminative power of the full BERT model. Its ability to closely approximate the ROC-AUC score of BERT highlights its suitability for scenarios where computational efficiency is as important as model performance, providing a balanced solution without significant compromises on capability. This can be seen in Figure 11.

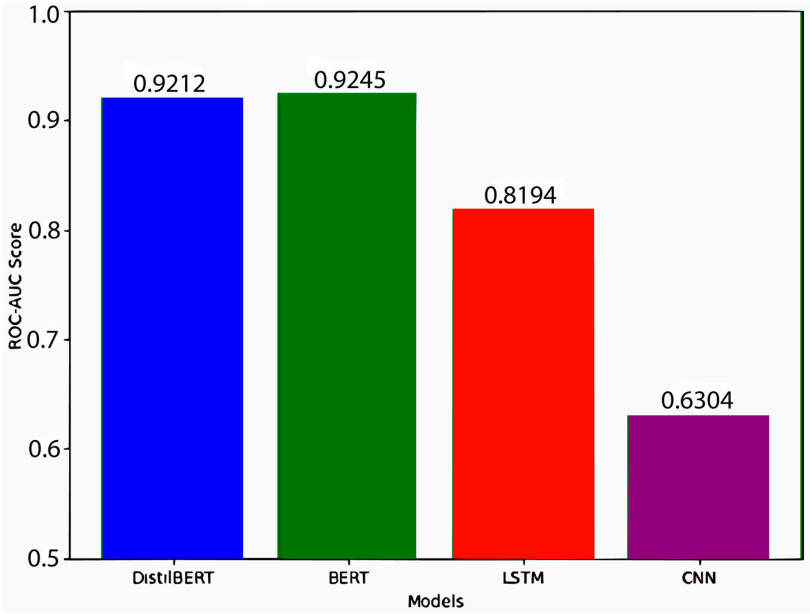


Figure 11. Comparison of ROC-AUC scores across the models.

Conversely, the ANN models show significantly lower ROC-AUC scores, indicating their weaker performance in effectively separating the classes (Table 9). These lower scores reflect the inherent limitations of these models in handling the complexity or specific characteristics of the dataset used in this study. LSTM, which is typically strong in sequential data analysis, and CNNs, which are effective in spatial data recognition, may not align well with the demands of tasks requiring intricate class separability. This could lead to higher

rates of false positives and false negatives, making these models less reliable compared to LLMs.

Table 9. Comparison of ROC-AUC scores of the CNN and LSTM models.

Model/Epoch	2	3	4	5	6	7	8	9
CNN	0.5617	0.5629	0.5782	0.6304	0.6293	0.6434	0.5615	0.6401
LSTM	0.8114	0.7665	0.8198	0.8194	0.7874	0.8024	0.7937	0.7986

Overall, among the models evaluated, BERT-based LLMs consistently showed high performance across all major metrics, particularly in maintaining strong accuracy, precision, recall, and F1-scores in later epochs. Their ability to sustain a low training loss without significant overfitting and achieve the highest ROC-AUC score suggests that these models not only learn effectively but also generalize well on unseen data. On the other hand, traditional ANNs underperformed relative to BERT-based LLMs. LSTM showed moderate performance in some metrics, but it did not compete on the same level in terms of accuracy or loss reduction. CNN, despite its simplicity and speed, failed to achieve competitive scores in any metrics other than a moderate improvement in accuracy and precision in the middle epochs.

BERT achieved the highest ROC-AUC score, demonstrating its exceptional ability to distinguish between classes; its high recall and precision confirm its effectiveness in identifying relevant instances while minimizing false positives. This performance can be attributed to its transformer-based architecture, which captures semantic relationships and handles the imbalanced class distribution effectively.

6.4. Main Findings of Experiment (1)

- BERT-based models outperform traditional architectures across the chosen key metrics. BERT demonstrated strong generalization and minimal overfitting, whereas DistilBERT performed well but was less stable, while LSTM and CNN underperformed, highlighting their limitations for complex tasks;
- A batch size of eight outperformed a batch size of sixteen, and Bayesian tuning (Optuna) improved the performance, highlighting the importance of fine-tuning parameters;
- BERT showed the most consistent loss reduction, indicating efficient learning and minimal overfitting. DistilBERT performed well but lagged slightly, while LSTM and CNN struggled with higher losses and slower optimization;
- BERT's highest ROC-AUC (92.45%) showed excellent class separability, while DistilBERT offered a competitive, efficient alternative. LSTM and CNN performed poorly in comparison.

7. Results, Analysis, Interpretation, and Findings of Experiment (2)

The classification performance of three machine learning models—LR, an SVM, and XGBoost—was assessed using precision, recall, and F1-score, providing a comprehensive view of how well each model identified and distinguished the security categories.

Regarding the ability of a model to make accurate predictions with minimal false positives (Figure 12), XGBoost demonstrated notable strengths, achieving high precision scores in categories such as "Data Protection and Encryption" (0.95) and "Threats, Attacks, and Intrusion" (0.86). These results reflect its ability to classify instances accurately in these categories. Logistic regression also performed well in the "Compliance, Trust, and Standards" category, achieving a precision of 0.86, indicating relatively strong predictive accuracy. The SVM outperformed the other models in categories such as "Compliance, Trust, and Standards" and "System Failures and Service Availability", achieving near-perfect pre-

cision. However, XGBoost struggled in the "Unwanted Communications" category, where it failed to identify any instances correctly, reflecting inconsistent precision performance in certain cases.

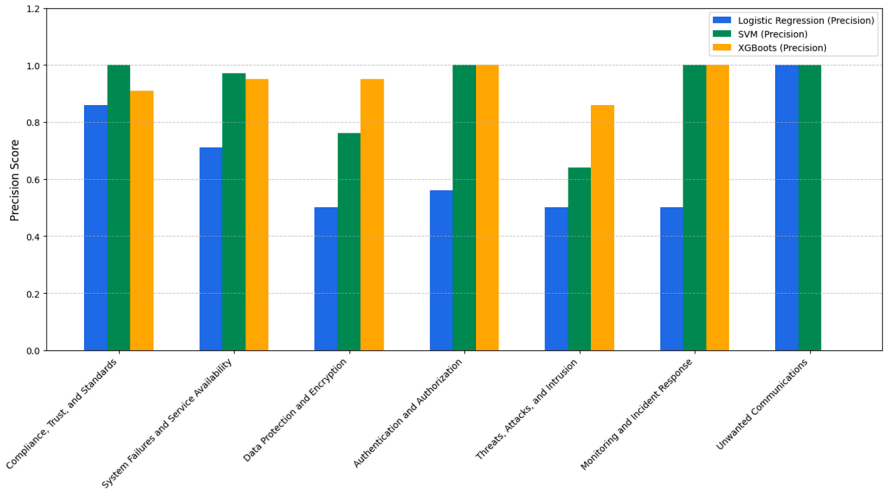


Figure 12. Comparison of precision scores of LR, the SVM, and XGBoost across security categories.

Furthermore, for evaluating a model's ability to identify all relevant instances by minimizing false negatives (Figure 13), the SVM achieved excellent recall in categories such as "Authentication and Authorization", where it achieved a perfect score of 1.00, as well as "System Failures and Service Availability" (0.90). Logistic regression showed strong recall in "Authentication and Authorization" (1.00) and "Threats, Attacks, and Intrusion" (1.00) but struggled in categories such as "Data Protection and Encryption" and "Monitoring and Incident Response", where its recall dropped to 0.50 and 0.80, respectively. XGBoost, while high in precision, exhibited inconsistent recall, particularly in the "Unwanted Communications" category, where it failed entirely. This limitation suggests that XGBoost may miss relevant instances in certain categories, indicating challenges in handling imbalanced or limited data.

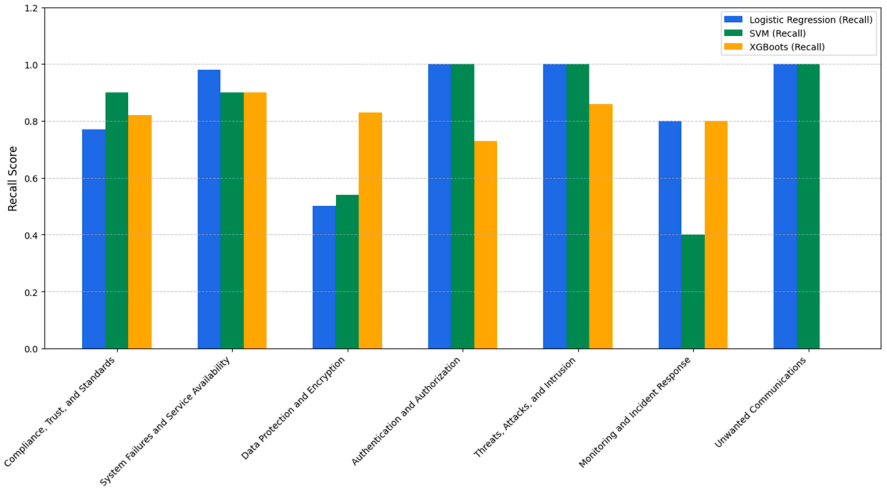


Figure 13. Comparison of recall scores of LR, the SVM, and XGBoost across security categories.

Moreover, measuring the F1-score, which balances precision and recall, provides a comprehensive measure of overall performance (Figure 14). The SVM emerged as the strongest performer, consistently achieving high F1-scores across most categories; it achieved 0.95 for "Compliance, Trust, and Standards", 0.94 for "System Failures and Service Availability", and a perfect 1.00 for "Authentication and Authorization".

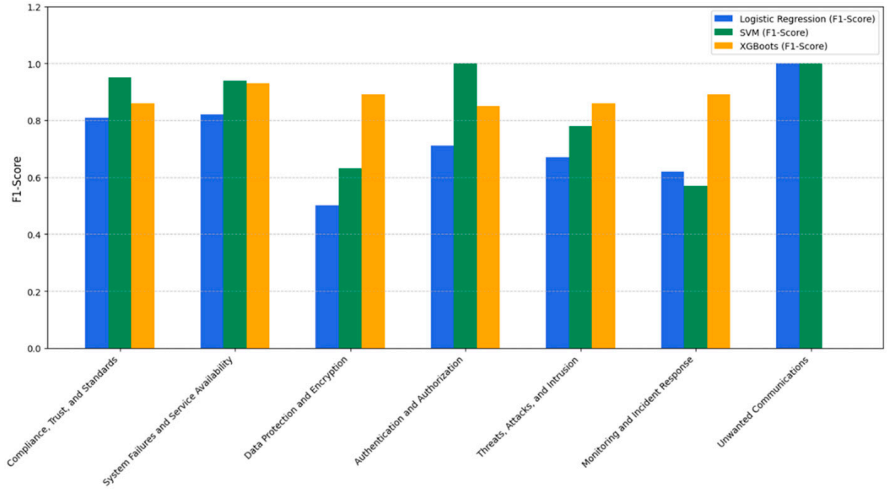


Figure 14. Comparison of F1-scores of LR, the SVM, and XGBoost across security categories.

However, the model struggled in "Monitoring and Incident Response", where it achieved an F1-score of only 0.57, primarily due to low recall (0.40). Logistic regression demonstrated moderate F1-scores, excelling in "Compliance, Trust, and Standards" (0.81) but underperforming in "Data Protection and Encryption" (0.50) and "Monitoring and Incident Response" (0.62). XGBoost delivered competitive F1-scores in categories such as "System Failures and Service Availability" (0.93) and "Data Protection and Encryption" (0.89), but it failed entirely in the "Unwanted Communications" category (0.00), significantly impacting its reliability.

In summary, the SVM demonstrated the best overall balance across precision, recall, and F1-score, making it the strongest candidate for security NFR classification. While logistic regression offered reasonable performance in some categories, its inability to handle more nuanced and complex categories limited its effectiveness. XGBoost excelled in precision and specific categories but suffered from inconsistent recall, particularly in cases with limited data. These results indicate that the SVM is the most robust and reliable model for this task, although further optimization may be necessary to address its recall limitations in certain categories.

Main Findings of Experiment (2)

- SVM achieved the best balance across metrics, excelling in categories such as "Authentication and Authorization" (F1: 1.00) and "Compliance, Trust, and Standards" (F1: 0.95), but struggled in "Monitoring and Incident Response" (F1: 0.57);
- XGBoost performed extremely well in precision (e.g., 0.95 in "Data Protection") but struggled with recall, failing entirely in "Unwanted Communications", highlighting issues with imbalanced data;
- LR performed well in "Compliance, Trust, and Standards" (F1: 0.81) but struggled in complex categories such as "Data Protection" (F1: 0.50), limiting its reliability for nuanced tasks.

8. Conclusions

This paper presents a new approach for assessing the security considerations of a software system in the requirements engineering phase. The proposed security framework addresses a wide range of cybersecurity challenges and dangers related to software systems, along with their corresponding techniques for avoiding risks at the early stages of development. Furthermore, in contrast to existing ML classification approaches, our

research has demonstrated that LLMs are effective at comprehending the semantics of security-related NFR phrases and accurately categorizing them into their respective classes.

In addition, a pre-trained and fine-tuned BERT model was created to classify software non-functional requirements into security NFRs and non-security NFRs based on the semantics of their statements. This model utilizes transfer learning and is a crucial part of the proposed approach for evaluating the security level of the software systems discussed in this research. Then, the proposed approach provides a deeper analysis of the security-related NFRs, classifying them into multiple predefined categories using a combined sentence-BERT embedding technique with an SVM. This hybrid model and the data augmentation technique (T5) applied to our dataset demonstrate the ability to tackle the small size issue in the utilized dataset and provide good results.

This research evaluated the effectiveness of the proposed approach through two experiments focused on binary and multi-class classification tasks. The first experiment assessed the performance of BERT-based language models and artificial neural networks (ANNs) in distinguishing between security and non-security NFRs. The results demonstrated the effectiveness of the BERT models in achieving high classification performance, highlighting their ability to generalize effectively across variations in NFR phrasing.

The second experiment focused on the multi-class classification of security NFRs into seven predefined categories using sentence-BERT embeddings with ML models, including LR, SVM, and XGBoost. These models were evaluated using standard metrics to capture their ability to minimize false positives, identify relevant instances, and balance precision, recall, and F1-scores across categories. The findings highlight the strengths and limitations of each model. Overall, these findings provide valuable insights for automating NFR classification processes, enhancing software security requirements elicitation, and supporting decision-making in software engineering contexts.

Future work should focus on creating a benchmark dataset specifically tailored for security-related NFR classification at the requirements engineering phase, enabling more comprehensive comparisons with related approaches and improving the generalizability of the proposed proof-of-concept framework.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available upon request; further inquiries can be directed to the corresponding author.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ML	Machine learning
LR	Logistic regression
SVM	Support vector machine
SRF	Security requirements fulfillment
NFR	Non-functional requirement
BERT	Bidirectional encoder representations from a transformer
ANN	Artificial neural network
CNN	Convolutional neural network
LSTM	Long short-term memory
LLM	Large language model

References

1. Saeed, S.; Altamimi, S.A.; Alkayyal, N.A.; Alshehri, E.; Alabbad, D.A. Digital Transformation and Cybersecurity Challenges for Businesses Resilience: Issues and Recommendations. *Sensors* **2023**, *23*, 6666. [CrossRef] [PubMed]
2. Herrmann, P.; Herrmann, G. Security requirement analysis of business processes. *Electron. Comm. Res.* **2006**, *6*, 305–335. [CrossRef]
3. Olivier, M.S. Database privacy: Balancing confidentiality, integrity and availability. *ACM SIGKDD Explor. Newslett.* **2002**, *4*, 20–27. [CrossRef]
4. Condori-Fernandez, N.; Lago, P. Characterizing the contribution of quality requirements to software sustainability. *J. Syst. Softw.* **2018**, *137*, 289–305. [CrossRef]
5. Binkhonain, M.; Zhao, L. A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert. Syst. Appl. X* **2019**, *1*, 100001.
6. Canedo, E.D.; Mendes, B.C. Software requirements classification using machine learning algorithms. *Entropy* **2020**, *22*, 1057. [CrossRef] [PubMed]
7. Baker, C.; Deng, L.; Chakraborty, S.; Dehlinger, J. Automatic multi-class non-functional software requirements classification using neural networks. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; pp. 610–615.
8. Sun, X.; Li, L.; Bo, L.; Wu, X.; Wei, Y.; Li, B. Automatic software vulnerability classification by extracting vulnerability triggers. *J. Softw. Evol. Process* **2024**, *36*, e2508. [CrossRef]
9. Marali, M.; Balakrishnan, K. Vulnerability Classification Based on Fine-Tuned BERT and Deep Neural Network Approaches. In *International Conference on Intelligent Systems and Sustainable Computing*; Springer Nature: Singapore, 2022; pp. 257–268.
10. Purba, M.D.; Ghosh, A.; Radford, B.J.; Chu, B. Software vulnerability detection using large language models. In Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), Florence, Italy, 9–12 October 2023; pp. 112–119.
11. Pan, C.; Lu, M.; Xu, B. An empirical study on software defect prediction using codebert model. *Appl. Sci.* **2021**, *11*, 4793. [CrossRef]
12. Das, S.S.; Serra, E.; Halappanavar, M.; Pothen, A.; Al-Shaer, E. V2w-bert: A framework for effective hierarchical multiclass classification of software vulnerabilities. In Proceedings of the 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), Porto, Portugal, 6–9 October 2021; pp. 1–12.
13. Ni, X.; Zheng, J.; Guo, Y.; Jin, X.; Li, L. Predicting severity of software vulnerability based on BERT-CNN. In Proceedings of the 2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI), Shijiazhuang, China, 22–24 July 2022; pp. 711–715.
14. Hu, P.; Tanchak, R.; Wang, Q. Developing risk assessment framework for wildfire in the United States–A deep learning approach to safety and sustainability. *J. Saf. Sustain.* **2024**, *1*, 26–41. [CrossRef]
15. Hao, Z.; Di Maio, F.; Zio, E. Monte carlo tree search-based deep reinforcement learning for flexible operation & maintenance optimization of a nuclear power plant. *J. Saf. Sustain.* **2024**, *1*, 4–13.
16. Ansari, M.T.L.; Pandey, D.; Alenezi, M. STORE: Security threat oriented requirements engineering methodology. *J. King Saud. Univ-Comput. Inf. Sci.* **2022**, *34*, 191–203. [CrossRef]
17. Ferraris, D.; Fernandez-Gago, C. TrUStAPIS: A trust requirements elicitation method for IoT. *Int. J. Inf. Secur.* **2020**, *19*, 111–127. [CrossRef]
18. Kang, K.C.; Cohen, S.G.; Hess, J.A.; Novak, W.E.; Peterson, A.S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*; Technical Report CMU/SEI-90-TR-21; Software Engineering Institute, Carnegie Mellon University: Pittsburgh, PA, USA, 1990. Available online: https://insights.sei.cmu.edu/library/feature-oriented-domain-analysis-foda-feasibility-study/ (accessed on 8 December 2023).
19. de Almeida, M.G.; Canedo, E.D. Authentication and authorization in microservices architecture: A systematic literature review. *Appl. Sci.* **2022**, *12*, 3023. [CrossRef]
20. Rupa, C.; Greeshmanth; Shah, M.A. Novel secure data protection scheme using Martino homomorphic encryption. *J. Cloud Comput.* **2023**, *12*, 47. [CrossRef]
21. Paliszkiewicz, J. Information security policy compliance: Leadership and trust. *J. Comput. Inf. Syst.* **2019**, *59*, 211–217. [CrossRef]
22. González-Granadillo, G.; González-Zarzosa, S.; Diaz, R. Security information and event management (SIEM): Analysis, trends, and usage in critical infrastructures. *Sensors* **2021**, *21*, 4759. [CrossRef]
23. Hindy, H.; Brosset, D.; Bayne, E.; Seeam, A.K.; Tachtatzis, C.; Atkinson, R.; Bellekens, X. A taxonomy of network threats and the effect of current datasets on intrusion detection systems. *IEEE Access* **2020**, *8*, 104650–104675. [CrossRef]
24. Tabrizchi, H.; Rafsanjani, M.K. A survey on security challenges in cloud computing: Issues, threats, and solutions. *J. Supercomput.* **2020**, *76*, 9493–9532. [CrossRef]
25. Makkar, A.; Garg, S.; Kumar, N.; Hossain, M.S.; Ghoneim, A.; Alrashoud, M. An efficient spam detection technique for IoT devices using machine learning. *IEEE Trans. Ind. Inform.* **2020**, *17*, 903–912. [CrossRef]

26. Navarro-Almanza, R.; Juarez-Ramirez, R.; Licea, G. Towards supporting software engineering using deep learning: A case of software requirements classification. In Proceedings of the 5th International Conference on Software Engineering Research and Innovation (CONISOFT), Mérida, Mexico, 25–27 October 2017; pp. 116–120.
27. Lima, M.; Valle, V.; Costa, E.; Lira, F.; Gadelha, B. Software engineering repositories: Expanding the PROMISE database. In Proceedings of the 33rd Brazilian Symposium on Software Engineering, Salvador, Brazil, 23–27 September 2019; pp. 427–436.
28. Subahi, A.F. BERT-Based Approach for Greening Software Requirements Engineering Through Non-Functional Requirements. *IEEE Access* **2023**, *11*, 103001–103013. [CrossRef]
29. Abad, Z.S.H.; Karras, O.; Ghazi, P.; Glinz, M.; Ruhe, G.; Schneider, K. What works better? A study of classifying requirements. In Proceedings of the IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, 4–8 September 2017; pp. 496–501. [CrossRef]
30. Jindal, R.; Malhotra, R.; Jain, A. Automated classification of security requirements. In Proceedings of the International Conference on Advanced Computing, Communication and Information (ICACCI), Jaipur, India, 21–24 September 2016; pp. 2027–2033. [CrossRef]
31. Severin, K.; Gokhale, S.; Dagnino, A. Keyword-based semi-supervised text classification. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; pp. 417–422.
32. Panagides, R.K.; Fu, S.H.; Jung, S.H.; Singh, A.; Eluvathingal Muttikkal, R.T.; Broad, R.M.; Meakem, T.D.; Hamilton, R.A. Enhancing Literature Review Efficiency: A Case Study on Using Fine-Tuned BERT for Classifying Focused Ultrasound-Related Articles. *AI* **2024**, *5*, 1670–1683. [CrossRef]
33. Hey, T.; Keim, J.; Koziolek, A.; Tichy, W.F. NoRBERT: Transfer learning for requirements classification. In Proceedings of the IEEE 28th International Requirements Engineering Conference (RE), Zurich, Switzerland, 31 August–4 September 2020; pp. 169–179. [CrossRef]
34. Barbon, R.S.; Akabane, A.T. Towards Transfer Learning Techniques—BERT, DistilBERT, BERTimbau, and DistilBERTimbau for Automatic Text Classification from Different Languages: A Case Study. *Sensors* **2022**, *22*, 8184. [CrossRef] [PubMed]
35. Sanh, V.; Debut, L.; Chaumond, J.; Wolf, T. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In Proceedings of the 5th NeurIPS EMC2 Workshop, Vancouver, BC, Canada, 13 December 2019.
36. Büyüköz, B.; Hürriyetoğlu, A.; Özgür, A. Analyzing ELMo and DistilBERT on socio-political news classification. In Proceedings of the Workshop on Automated Extraction of Socio-political Events from News 2020, European Language Resources Association (ELRA), Marseille, France, 11–16 May 2020; pp. 9–18.
37. Maragheh, H.K.; Gharehchopogh, F.S.; Majidzadeh, K.; Sangar, A.B. A Hybrid Model Based on Convolutional Neural Network and Long Short-Term Memory for Multi-label Text Classification. *Neural Process. Lett.* **2024**, *56*, 42. [CrossRef]
38. Shambharkar, P.G.; Singh, H.; Raghav, H.R.; Verma, H. Exploring the Efficacy of Deep Learning Models for Multiclass Toxic Comment Classification in Social Media Using Natural Language Processing. In Proceedings of the 2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 25–26 May 2023; pp. 1–8.
39. Le, T.T.; Oktian, Y.E.; Kim, H. XGBoost for imbalanced multiclass classification-based industrial internet of things intrusion detection systems. *Sustainability* **2022**, *14*, 8707. [CrossRef]
40. Morales-Hernández, R.C.; Jagüey, J.G.; Becerra-Alonso, D. A Comparison of Multi-Label Text Classification Models in Research Articles Labeled with Sustainable Development Goals. *IEEE Access* **2022**, *10*, 123534–123548. [CrossRef]
41. Kenton, J.D.; Toutanova, L.K. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, 2–7 June 2019; Association for Computational Linguistics: Kerrville, TX, USA, 2019; Volume 1, pp. 4171–4186.
42. Loshchilov, I.; Hutter, F. Fixing weight decay regularization in adam. *arXiv* **2017**, arXiv:1711.05101.
43. Liu, Y. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
44. Zhou, C.; Sun, C.; Liu, Z.; Lau, F. A C-LSTM neural network for text classification. *arXiv* **2015**, arXiv:1511.08630.
45. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning, JMLR.org, Atlanta, GA, USA, 16–21 June 2013; Volume 28, pp. 1310–1318.
46. Kingma, D.P.; Ba, L.J. Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015; ArXiv: Ithaca, NY, USA, 2015; pp. 1–13. Available online: https://dare.uva.nl/search?identifier=a20791d3-1aff-464a-8544-268383c33a75 (accessed on 28 December 2024).
47. Rebuffi, S.A.; Gowal, S.; Calian, D.A.; Stimberg, F.; Wiles, O.; Mann, T.A. Data augmentation can improve robustness. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 29935–29948.
48. Abonizio, H.Q.; Junior, S.B. Pre-trained data augmentation for text classification. In *Brazilian Conference on Intelligent Systems*; Springer International Publishing: Cham, Switzerland, 2020; pp. 551–565.
49. Abonizio, H.Q.; Paraiso, E.C.; Barbon, S. Toward text data augmentation for sentiment analysis. *IEEE Trans. Artif. Intell.* **2021**, *3*, 657–668. [CrossRef]

50. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* **2020**, *21*, 140.
51. Zheng, W.; Cheng, J.; Wu, X.; Sun, R.; Wang, X.; Sun, X. Domain knowledge-based security bug reports prediction. *Knowl.-Based Syst.* **2022**, *241*, 108293. [CrossRef]
52. Subahi, A.F.; Athama, A. Fuzzy Logic Inference System for Managing Intensive Care Unit Resources Based on Knowledge Graph. *Comput. Mater. Contin.* **2023**, *77*, 3801–3816. [CrossRef]
53. Piao, G. Scholarly text classification with sentence BERT and entity embeddings. In Proceedings of the Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2021 Workshops, WSPA, MLMEIN, SDPRA, DARAI, and AI4EPT, Delhi, India, 11 May 2021; Springer International Publishing: Berlin/Heidelberg, Germany, 2021; pp. 79–87.
54. Chu, Y.; Cao, H.; Diao, Y.; Lin, H. Refined SBERT: Representing sentence BERT in manifold space. *Neurocomputing* **2023**, *555*, 126453. [CrossRef]
55. Muralidhar, R.; Demory, M.L.; Kesselman, M.M. Exploring the Impact of Batch Size on Deep Learning Artificial Intelligence Models for Malaria Detection. *Cureus* **2024**, *16*, e60224. [CrossRef]
56. Nado, Z.; Gilmer, J.M.; Shallue, C.J.; Anil, R.; Dahl, G.E. A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes. *arXiv* **2021**, arXiv:2102.06356.
57. Afaq, S.; Rao, S. Significance Of Epochs On Training A Neural Network. *Int. J. Sci. Technol. Res.* **2020**, *9*, 485–488.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.