AN EXPLORATORY EXAMINATION OF SOFTWARE VULNERABILITY CLASSIFICATION USING LARGE LANGUAGE MODELS

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAIʻI AT MĀNOA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

MAY 2024

By

Ana Oliveira Araujo

Thesis Committee:

Dr. Anthony Peruma, Chairperson
Dr. Mehdi Tarrit Mirakhorli
Dr. Rick Kazman

Keywords: cybersecurity, software vulnerability, CVE, VDO, vulnerability classification, LLMs.

ABSTRACT

Software vulnerabilities are critical weaknesses that can compromise the security of a system. While current research primarily focuses on automating the classification and detection of them using a range of machine learning models, there remains a notable gap in integrating ontologies like the Vulnerability Description Ontology with Large Language Models (LLMs) for enhanced classification accuracy. Our study utilizes the National Vulnerability Database (NVD) and the National Institute of Standards and Technology's Vulnerability Description Ontology framework to enhance the classification of these vulnerabilities. The methodology involves an in-depth analysis of NVD data and an investigation of the effectiveness of various LLMs to analyze vulnerability descriptions across 27 vulnerability categories in 5 noun groups. Our findings reveal that LLMs, particularly BERT and DistilBERT, demonstrate stronger performance when compared to traditional machine learning models and entropy-based methods. Moreover, while expanding the dataset aims to capture a broader range of vulnerabilities, its effectiveness varies, highlighting the crucial role of annotation quality. This research emphasizes the importance of advanced machine learning techniques and quality data annotation in optimizing vulnerability assessment processes in cybersecurity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Advancements in technology have transformed operations across numerous industries, bringing substantial benefits but also introducing new risks. As organizations increasingly depend on digital solutions, the importance of safeguarding these systems has become fundamental. One of the most pressing issues in this context is the prevalence of software vulnerabilities. These vulnerabilities are not mere glitches; they are significant weaknesses that, if exploited by malicious agents, can allow unauthorized access to or modification of sensitive data. This exploitation can compromise the integrity, availability, and confidentiality of information, leading to severe security risks with potentially extensive economic and reputational consequences.

For instance, in November 2023, Romanian cybersecurity firm Bitdefender uncovered multiple vulnerabilities in LG webOS that powers its smart televisions. These vulnerabilities, identified as CVE-2023-6317 through CVE-2023-6320, allowed attackers to bypass authorization mechanisms and gain root access on the devices. One of the flaws (CVE-2023-6318) enabled attackers to escalate their privileges, potentially taking full control of the TV. Another vulnerability (CVE-2023-6319) involved operating system command injection through a component managing music lyrics, while CVE-2023-6320 allowed the injection of authenticated commands through a specific API endpoint. Such exploitation could give threat actors unprecedented control over the TVs without any user interaction, posing significant privacy and security risks. These vulnerabilities were subsequently patched by LG in a series of updates released on March 22, 2024 [28].

In recent years, the prevalence of software vulnerabilities have escalated, underscoring the critical need for effective management and analysis of these threats. Effectively classifying and managing vulnerabilities can significantly diminish the risk of attacks and potential damage to systems. For this reason, there has been a significant rise in the use of empirical methods to analyze and predict vulnerable components. These efforts primarily focus on addressing the following questions: can we predict software vulnerabilities, classify their types, and assess their severity?

However, if our data sources fail to fully capture the phenomenon of interest, our predictions may excel within the confines of our data but prove unsatisfactory in practice. Typically, researchers rely on public vulnerability databases like CVE or NVD, or vendor-specific databases, sometimes combining both. These databases employ manual processes to analyze vulnerabilities, reviewing text-based reports, advisories, and patch details. Nevertheless, it becomes time-consuming given the substantial number of vulnerabilities published annually. Consequently, this intensive process has led to issues regarding the quality of vulnerability reports, with studies indicating incompleteness and inconsistencies Ozmet [31].

Fortunately, recent advancements in machine learning offer potential solutions to automate vulnerability classification. For instance, studies by Russo [34], Gonzalez[14], and Okutan [29] have

showcased the effectiveness of employing various machine learning classifiers to automatically characterize software vulnerabilities based on CVE descriptions. Additionally, deep learning models, such as Huang's [18] TFI-DNN model, integrating Information Gain and Deep Neural Networks, have demonstrated superior performance over traditional methods in vulnerability detection and classification.

While these machine learning approaches have shown promise, the transformer-based approach also holds significant potential in vulnerability detection and classification, as seen in Steenhoek [36], Chandra [37], Gao [13], Fu [12], Omar [30].

This paper explores the application of large language models (LLMs) to the task of software vulnerability classification. Our approach utilize various Transformer architectures including BERT, DistilBERT, XLNet, DEBERTA V3, and LLAMA 2 to classify attribute characteristics based on the Vulnerability Description Ontology (VDO)established by the National Institute of Standards and Technology (NIST) Booth [4]. The study is structured around three primary research questions that aim to dissect model and data expansion effectiveness, and linguistic features relevant to vulnerability classification using LLMs.

Research Questions:

- **RQ1: Does the expansion of the dataset lead to enhanced performance in vulnerability classification by large language models?** The first question addresses whether extending the data can lead to stronger models capable of categorizing a wider array of vulnerability types.

- **RQ2: How do large language models impact the effectiveness of software vulnerability classification?** The second question seeks to evaluate the performance of LLMs in the software vulnerability classification domain. It encompasses two sub-questions:

  - **RQ2.1: How do models perform when classifying 27 vulnerability attributes?** This sub-question aims to compare the different models, transform-based vs. traditional, in accurately classifying software vulnerability attributes.
  - **RQ2.2: Which models have the best average f1 score across 27 vulnerability attributes?** This sub-question aims to identify which individual models demonstrate the greatest overall effectiveness.

- **RQ3: What are the prevalent n-grams associated with each vulnerability attribute?** The third question shifts focus to the linguistic aspects, exploring the specific n-grams that are most prevalent and potentially indicative of different types of software vulnerabilities.

Contributions:

This research work contributes to the field of cybersecurity and machine learning by providing empirical evidence on the capabilities of large language models (LLMs) in enhancing software vulnerability classification. It systematically evaluates the capacity of LLMs and compares their effectiveness against traditional models. Additionally, it explores the impact of data extension on model performance and identifies key linguistic features used by these models. The findings offer valuable insights for security professionals, indicating that LLMs can lead to more accurate assessments and reduce response time for mitigation strategies. To support further research and replication of the results, the source code and annotated data used in this study are made available on GitHub [2]. Moreover, the research underscores the importance of quality control in data annotation, highlighting its significance in improving model performance.

The paper's structure is as follows: Section II offers an explanation of the Vulnerability Description Ontology, the National Vulnerability Database, and transformer-based models. Section III gives an overview of related work in the field. Section IV gives a detailed presentation of our proposed vulnerability classification methodology, including the process of data annotation, relevant algorithms and performance metrics. Section V presents the experimental results, and Section VI addresses the threats of validity. Finally, Section VII concludes the paper.

CHAPTER 2
BACKGROUND

2.1 Security Vulnerabilities

In the domain of cybersecurity, a security vulnerability is formally described as a flaw or weakness present in software that allows for potential exploitation by attackers to gain unauthorized access to a system or network, as per the definition provided by the Common Vulnerability Exposures (CVE) terminology Booth [3]. These vulnerabilities typically manifest as unexpected software behaviors or as a result of inadequate security measures. Due to the high risk associated with such vulnerabilities, they are deemed to be of great importance, with their identification and remediation often taking precedence over other software deficiencies. In response to the discovery of vulnerabilities, software vendors are compelled to issue a release of patches or new software versions to mitigate the risks and minimize the potential impact.

2.2 National Vulnerability Database (NVD)

To facilitate the dissemination of information regarding security vulnerabilities and to support the development of secure software, such vulnerabilities are reported in databases that are accessible to the public. A preeminent example of such a repository is the National Vulnerability Database (NVD) Booth [3], which was introduced by the National Institute of Standards and Technology (NIST) with the intention of promoting secure software development practices, as well as to streamline the public disclosure and management of software vulnerabilities.

The NVD is an reliable repository that has been comprehensively recording software vulnerabilities since 2001. The scope of the NVD encompasses a diverse array of 36,436 products, spanning categories such as applications, operating systems, and hardware Booth [3]. Fundamentally, the NVD is predicated on the CVE List — a collection that enumerates vulnerabilities, each entry comprising a unique identifier, a description, and at least one publicly accessible reference. This identifier, known as the Common Vulnerability Exposures (CVE) number or ID, serves as a standardized reference point for every publicly disclosed vulnerability.

The NVD augments the CVE entries with additional critical details such as the quantified severity, encapsulated in the Common Vulnerability Scoring System (CVSS), and the categorical type, referred to as the Common Weakness Enumeration (CWE) as illustrated in Fig. 2.1. Each entry in the database is evaluated using the Common Vulnerability Scoring System (CVSS), which is a standardized framework for rating the severity of security vulnerabilities. The CVSS allocates scores based on a set of criteria that includes Access Complexity, Authentication, Access Vector, Confidentiality Impact, Integrity Impact, and Availability Impact. These criteria are merged to

derive the Impact, Exploitability, and Base Scores, providing a quantifiable measure of vulnerability severity. Despite its expansive coverage, it is worthy to mention that the NVD does contain a minimal margin of error in its records Ozmet [31].

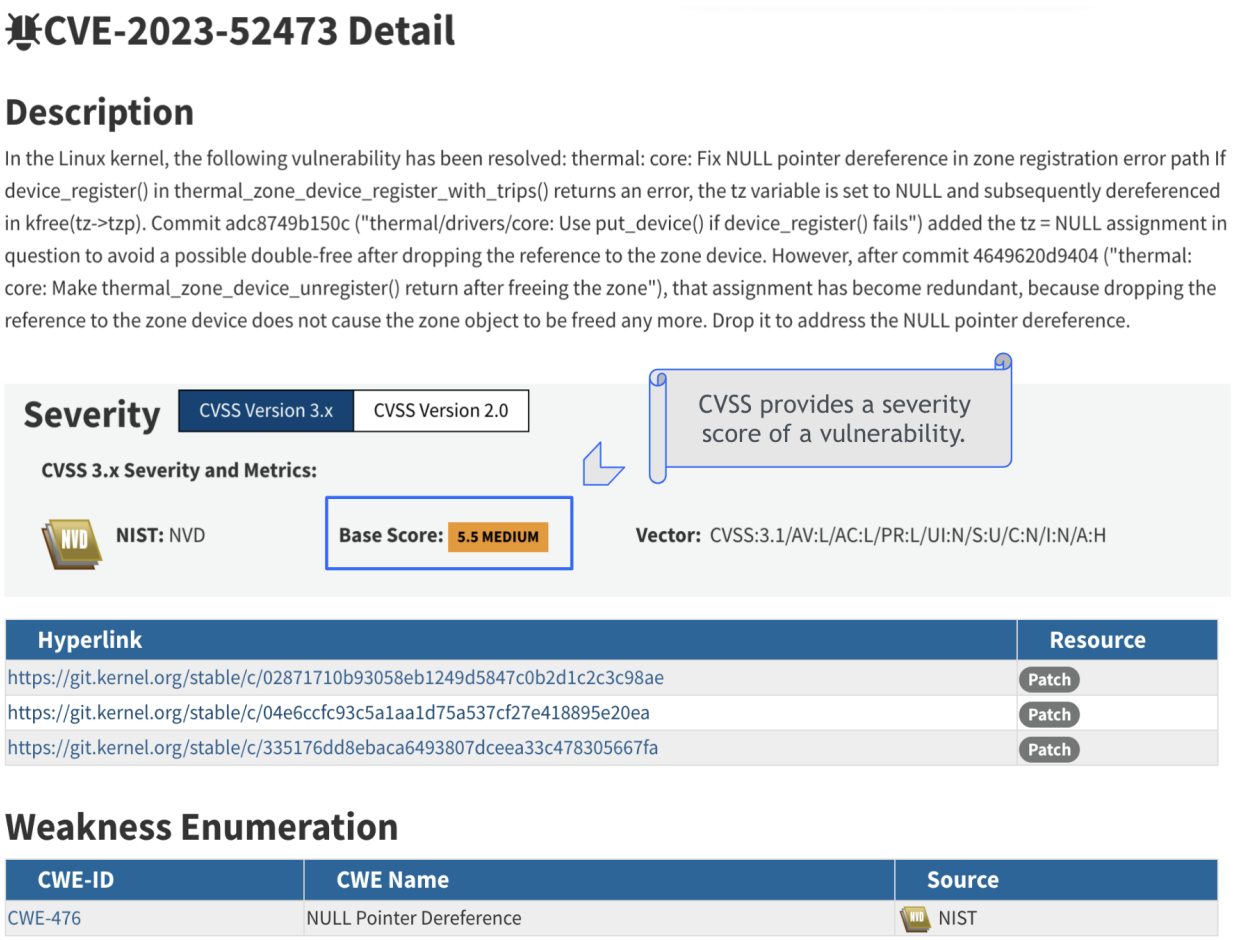

Figure 2.1: CVEs critical details employed by the National Vulnerability Dataset, e.g., CVE-2023-52473.

2.3 Vulnerability Description Ontology (VDO)

Understanding the nature of software vulnerabilities facilitates the identification of foundational issues, enables a comprehensive assessment of potential consequences, and guides the development of mitigation strategies. In support of this effort, the National Institute of Standards and Technology (NIST) has introduced a standardized Vulnerability Description Ontology (VDO) to aid in the process of characterization of software vulnerabilities [4]. In this study, we analyzed five noun groups from the Vulnerability Description Ontology (VDO). The selection of these noun groups — Attack Theater, Context, Impact Method, Logical Impact, and Mitigation — was grounded in

their relevance to the comprehensive characterization of vulnerabilities and their significance within cybersecurity research.

This investigation contributes to the existing body of knowledge by offering a detailed overview of the VDO's utility in CVE analysis, as presented in Table 2.3. This framework not only enhances the theoretical understanding of vulnerabilities but also serves as a practical guide for developing effective cybersecurity measures. For example, the noun group Attack Theater points to potential locations for cybersecurity breaches. Context emphasizes the environments or circumstances where the impacts of vulnerabilities manifest. Through Impact Method, we examine the array of tactics that attackers might deploy. Logical Impact provides insight into the possible consequences of attacks. Meanwhile, Mitigation focuses on strategies to either prevent or lessen the effects of successful exploits.

2.4 Language Models

Language models (LMs) serve as foundational tools for a wide array of tasks in natural language processing (NLP), including but not limited to machine translation, sentiment analysis, question answering, and text summarization. The paper "Attention is All You Need" by Vaswani [39] introduced the Transformer model, revolutionizing the way language models process and understand text through self-attention mechanisms.

2.4.1 BERT (2018)

The introduction of BERT, which stands for Bidirectional Encoder Representations from Transformers, in 2018, paved the way for a profound transformation in the realm of language representation models. According to Devlin [11], this method conditions on both left and right contexts across all layers through the use of bidirectional self-attention. The traditional language models prior to BERT missed out on utilizing the context available on both sides of a word. This inherent limitation reduces their linguistic comprehension, failing to encapsulate context, integral to understanding natural language.

The BERT model, and much of Language Models, undergoes a two-step process: pre-training and fine-tuning. In the pre-training phase, BERT is pre-trained on a large corpus of unlabeled text across various tasks, allowing it to learn universal language patterns. In the fine-tuning phase, BERT adapts to specific tasks by fine-tuning its pre-trained parameters with task-specific labeled data. This customization allows for the creation of models tailored to a variety of tasks, such as sentiment analysis or entity recognition, without altering the foundational pre-trained parameters. This advancement not only facilitates a seamless transition from pre-training to fine-tuning without the need for major task-specific architectural changes, but also ensures the preservation of the model's language comprehension for an array of tasks, including text classification.

The pre-training involves two novel objectives: the masked language model (MLM) and the next sentence prediction (NSP). Specifically, the MLM task improves the model's contextual understanding by predicting randomly masked tokens based solely on their context, while the NSP task enhances its ability to discern relationships between sentences. This phase doesn't rely on task-specific data, enabling the model to generalize across a broad spectrum of language understanding tasks.

- Masked Language Model (MLM): The MLM approach aims to create a bidirectional comprehension of text by randomly masking certain input tokens and subsequently predicting these obscured tokens. In practice, it randomly obscures 15% of the WordPiece tokens in a sequence. For a selected token:

  - 80% of the time, it is replaced with the [MASK] token to obscure it directly.
  - 10% of the time, it is replaced with a random token, introducing noise and forcing the model to improve its contextual understanding.
  - The remaining 10% of the time, the token is left as it is, ensuring the model can still predict correctly even without any masking.

  Consider the phrase "*The cat sat on the mat.*" If "sat" is chosen for masking, the model engages in three potential alterations:

  - In 80% of instances, it transforms to "*The cat [MASK] on the mat,*" concealing the specific action.
  - There's a 10% chance it might change to "*The cat danced on the mat,*" substituting "*sat*" with an unrelated verb.
  - The remaining 10% possibility leaves the sentence unaltered as "*The cat sat on the mat,*" to ensure the model can predict the original word even without modifications.

- Next Sentence Prediction (NSP): As noted by Devlin [11], prior to BERT, traditional language models felt short in capturing the interplay between sentences. To bridge this gap, the NSP task, evaluates the model's ability to predict whether a given sentence B logically follows another sentence A. During pre-training, for every pair of sentences selected, there's an equal probability that sentence B either directly follows A or is a randomly chosen sentence from the dataset, labeled correspondingly as IsNext or NotNext.

  Example: In a scenario where sentence A states "*The sky is clear today,*" and sentence B is "*We should go to the beach,*" the model, leveraging NSP, determines the logical progression between these sentences as IsNext. Conversely, if sentence B were "*Elephants are the largest land animals,*" the model identifies the lack of coherence, tagging the pair as NotNext.

2.4.2 DistilBERT (2019)

Following the breakthrough of BERT, subsequent models sought to refine and expand upon these foundational techniques. DistilBERT (2019) [35], which is a distilled version of the BERT model, streamlined architecture simplifies BERT by omitting token-type embeddings and the pooler and halving the number of layers. It optimized the process by distilling BERT's capabilities into a smaller, more efficient model, thereby making advanced NLP technologies more accessible for real-world applications with limited computational resources. It manages to reduce the original BERT model's size by 40% and increase processing speed by 60%, all while retaining 97% of BERT's language understanding performance. Such models facilitate broader adoption by easing the computational load for training and inference.

2.4.3 XLNet (2019)

XLNet (2019) [42] innovated further by integrating the best aspects of autoregressive (AR) language modeling and autoencoding (AE), eliminating the limitations of BERT's MLM approach through permutation-based language modeling. This enhanced the model's ability to understand complex language patterns and contexts without relying on data corruption techniques. This is achieved through two ways:

1. Firstly, unlike conventional AR models that follow a fixed sequence for factorization, XLNet dynamically maximizes the expected log likelihood across all possible permutations of a sequence. This strategy enables each token to be contextualized using information from both preceding and succeeding tokens, thereby capturing context without the need for data corruption techniques like those used in BERT.

2. Secondly, XLNet's avoidance of data corruption means it does not suffer from the pre-train and finetuning discrepancy that plagues BERT, where artificial [MASK] tokens used during pre-training are absent during fine-tuning . Instead, XLNet employs an autoregressive mechanism to predict each token, thus respecting the joint probability of the sequence and eliminating the simplifying assumption of token independence inherent in BERT's approach.

In terms of the architecture, XLNet incorporates significant innovations from Transformer-XL, particularly the segment recurrence mechanism and relative positional encodings. Furthermore, it is tailored to model multiple segments in a coherent manner, mirroring BERT's handling of paired sequences but within an AR framework. XLNet has demonstrated superior performance compared to BERT [11] and ROBERTA [25] across a wide array of benchmarks, including question answering, sentiment analysis, and document ranking, according to Yang [42].

2.4.4 DeBERTa V3 (2021)

DeBERTa improves upon the traditional BERT model by introducing two innovative components: Disentangled Attention (DA) and an enhanced mask decoder. According to He [17], the DA mechanism uses two separate vectors for the content and position of each input word, enabling more precise attention weight computations based on both content and relative positions. DeBERTa also employs an enhanced mask decoder during its pre-training phase, which incorporates absolute position information to improve the effectiveness of masked language modeling (MLM). DeBERTaV3 (2021) [16] then combined the strengths of DeBERTa's disentangled attention with ELECTRA's [9] efficient RTD training method, further enhancing model performance with the introduction of Gradient-Disentangled Embedding Sharing (GDES). The experiments shows that DeBERTaV3 base with GDES outperforms state-of-the-art models up to 2021 - BERT, RoBERTA, XLNet, ELECTRA, DeBERTa, across a variety of natural language understanding tasks.

2.4.5 Llama 2 (2023)

In 2023, the evolution reached a new height with the introduction of Llama 2 [38]. Llama 2 distinguishes itself through advanced training strategies, such as leveraging more diverse and extensive datasets, which enrich its understanding of language nuances across various domains. Its architectural innovations provide enhanced processing of long-range dependencies, making it adept at comprehending and generating more coherent text. Moreover, Llama 2 implements more efficient attention mechanisms and optimization techniques, improving both the speed and accuracy of the model while reducing computational resource requirements.

Noun Group	Category	Definition
Attack Theater	Remote	Initiates attacks through network connections external to the target's local network, primarily via the Internet.
	Local	Requires attacker's logical access to the device through interfaces like console, RDP, SSH.
	Limited Remote	Conducts attacks over layer 2 or 3 technologies, constrained by network communication or distance, e.g., Cellular, Wireless.
	Physical	Demands attacker's physical presence at the device's location for execution.
Context	Application	Software developed to perform specific tasks, capable of running across operating systems, firmware, or within other applications.
	Hypervisor	Manages hardware resources for multiple operating systems, allowing each to operate with its own virtual resources while ensuring isolation and efficient resource allocation.
	Firmware	Built-in software embedded within devices like routers, firewalls, and BIOS/UEFI, providing fundamental operational instructions.
	Physical Hardware	Constitutes the tangible components of technology, including processor logic gates, disk sectors, or memory cells.
	Channel	Facilitates logical communication between entities, relevant in cases of inherent protocol or cipher suite flaws, like insufficient entropy.
	Host OS	Serves as the primary operating system, supporting software applications, applicable outside the hypervisor context; contrasts with Guest OS.
	Guest OS	Operates under a hypervisor, functioning as an isolated operating system for specific applications, used in contrast to Host OS.
Impact Method	Trust Failure	Enables an attacker to bypass sandbox security, moving from a restricted environment to another, exploiting trust mechanisms.
	Context Escape	Occurs when an exploit misuses assumed trust relationships, leading to unintended consequences, such as verification failures or trust assumption errors.
	Authentication Bypass	Allows attackers to gain access or permissions by failing to correctly identify them, undermining security measures.
	Man-in-the-Middle	Requires an attacker to intercept communications between two parties, exploiting trust to manipulate or eavesdrop on the exchange.
	Code Execution	Permits attackers to run unauthorized code, affecting the intended operational context and compromising system integrity.
Logical Impact	Service Interrupt	Enables attackers to disrupt service availability, either partially or completely, without authorization.
	Read	Allows attackers to breach confidentiality through unauthorized data access.
	Write	Enables attackers to compromise data integrity by unauthorized modification or data addition.
	Privilege Escalation	Grants attackers higher access levels than intended, potentially leading to comprehensive unauthorized actions.
	Resource Removal	Permits attackers to delete data without authorization, affecting resource integrity.
	Indirect Disclosure	Allows attackers to indirectly learn information about the system, utilizing methods that bypass direct data access.
Mitigation	ASLR	Implements Address Space Layout Randomization (ASLR) to enhance security by randomizing memory addresses.
	Multi-Factor Authentication	Requires Multi-Factor Authentication for access, adding a layer of security beyond passwords.
	Sandboxed	Operates the product within a sandbox, isolating it from other system components to limit potential breaches.
	HPKP/HSTS	Utilizes HTTP Public Key Pinning (HPKP) or HTTP Strict Transport Security (HSTS) to secure web communications.
	Physical Security	Employs physical security measures to mitigate vulnerabilities, protecting against unauthorized physical access.

Table 2.1: Vulnerability attribute domains (noun groups) in the VDO framework examined in this study.

CHAPTER 3
LITERATURE REVIEW

Our literature review embarks on a comprehensive journey through the methodologies deployed in the analysis and detection of software vulnerabilities, outlined in Table 3.1. It initiates with a look at the manual curation process, underscored by the investigative work of Jimenez [19] and Nguyen [26], which lays the groundwork by highlighting the critical importance of accurate data collection and labeling. This segment transitions into a discussion on the broader spectrum of machine learning techniques, showcasing how conventional methods have been applied to address the intricacies of software vulnerabilities.

Following the exploration of traditional machine learning strategies, we delve into the realm of deep learning (DL), where the focus shifts to more advanced, data-driven models capable of capturing and analyzing complex patterns within vast datasets. The review culminates in an examination of the transformative impact of transformer architectures, highlighting their groundbreaking contributions to the field of vulnerability prediction.

Through this structured overview, we aim to navigate the significant advancements in the field, from the foundational manual curation efforts to the cutting-edge implementations of deep learning and transformers, setting the stage for a detailed discussion on each method's contributions within the broader context of cybersecurity research.

3.1 Manual Annotation of Software Vulnerabilities

The works of Jimenez [19] and Nguyen [26] provide a comprehensive overview of the challenges and considerations inherent in the use of various data sources and predictive models for studying software vulnerabilities. Jimenez [19] challenges the prevailing assumption of perfect labeling in datasets. Their research, based on a dataset encompassing 1,898 real-world vulnerabilities from projects like the Linux Kernel, OpenSSL, and Wireshark, demonstrates the stark contrast in model effectiveness under realistic conditions. The results challenge the optimistic conclusions drawn from models trained under the unrealistic assumption of perfect labeling, showing a significant drop in predictive effectiveness when realistic labeling is considered.

Similarly, Nguyen's [26] empirical analysis on the effectiveness of different data sources for studying software vulnerabilities, particularly in Mozilla Firefox, sheds light on the critical aspect of data reliability and completeness. They underscores the value of choosing the right dataset by evaluating the information sourced from public vulnerability databases like the NVD, Mozilla's own advisories (MFSA), and Bugzilla reports. This choice directly impacts the quality of vulnerability studies, highlighting the need for more accurate data collection methods to improve the predictive capabilities of vulnerability models.

The critical insights provided by both Jimenez [19] and Nguyen [26] naturally segue into the

discussion on manual curation of software vulnerability datasets. The process of manually curating data not only addresses the challenges highlighted in these studies but also serves as a critical step in enhancing the accuracy and reliability of the datasets used for machine learning applications. This section synthesizes the methodological approaches to manual curation documented by researchers, exploring its fundamental role in the classification of software vulnerabilities.

Chen [8], Okutan [29], Gonzalez [14], Chen [7], Russo [34], and Zhou [44] collectively underscore the importance of manual curation in enhancing the quality of vulnerability datasets. The initial stage of manual curation involves the collection and review of potential vulnerability-related information from a variety of sources, including Jira tickets, Bugzilla reports, open-source projects, and CVE entries from the National Vulnerability Database (NVD). This extensive data gathering effort aims to compile comprehensive information from open-source libraries and software, which serves as the foundational dataset for subsequent analysis.

The rigor of the manual curation process is evident in the detailed labeling and annotation practices employed by researchers. For instance, Chen [8] describe a scenario where security researchers carefully review collected data to label them as either vulnerability-related or not. Similarly, Okutan [29] elaborate on an annotation process designed to ensure high-quality data labeling, where Common Vulnerabilities and Exposures (CVE) entries are assigned confidence scores by Subject Matter Experts (SMEs). These scores reflect the experts' certainty about the accuracy of the labels, with further scrutiny applied to CVEs marked with low confidence scores through peer-discussion sessions.

Gonzalez [14] manually curate a dataset of 365 vulnerability descriptions, each mapped to one of 19 characteristics from the NIST Vulnerability Description Ontology (VDO). This dataset is created through a peer review process, where numerous CVEs and their descriptions from the NVD were evaluated. These CVE reports and their characteristics are then reviewed by two security experts from the research group to ensure agreement with the labeling performed by the original developers before being included in the manually curated training dataset. Different from vulnerability descriptions, Zhou [44] utilized large-scale open-source C projects in their work with Devign, where the data was manually labeled from four diverse datasets. These datasets incorporated a variety of real source code complexities, aiming to overcome limitations of synthetic code used in previous studies. Chen [7] manually review 1,128 vulnerability reports through the use of the open card sorting method to categorize rejected and disputed CVEs, revealing the complexity and depth of manual analysis required to accurately characterize vulnerabilities.

As the field advances, the integration of manual curation with automated tools represents an evolving landscape, augmenting the precision and effectiveness of vulnerability management systems. In particular, the study by Russo [34] demonstrates the practical application of manual curation in validating automated tools within vulnerability management. Their study involved a manual review process utilizing a dataset of 3,369 CVE records, pre-labeled by industry experts

according to a specific vulnerability taxonomy, to validate the classification and summarization capabilities of their tool, CVErizer. The tool demonstrated high accuracy in classifying vulnerabilities and generating summaries deemed useful by cybersecurity experts. Furthermore, the research included an end-to-end evaluation with cybersecurity students and professional security experts to assess the usefulness of CVErizer-generated summaries in supporting vulnerability assessment activities. Likewise, Okutan's [29] system show the potential to evaluate vulnerabilities up to 95 hours faster than manual methods, characterize them with F-Measure values over 0.9, and achieve up to 47% time savings in CVE classification efforts.

Through these examples, it is clear that manual curation is not merely a preliminary step but a foundation of the vulnerability classification process. The expertise and judgment of security researchers are indispensable in refining the datasets that feed into machine learning models, ensuring that these models are built upon a foundation of accurately labeled and reviewed data.

3.2 Machine Learning Models

3.2.1 Conventional

Zhang [43] and Last [21] focus on predicting the time to the next vulnerability and vulnerability discovery rates, respectively, using regression models and K-NN classification. The K-NN classification is utilized to select the most appropriate regression model for forecasting, based on time series distance measurements. The results indicate challenges in achieving accurate forecasts due to the smoothness of the underlying data, suggesting that the success of forecasting models depends on consistent trends in vulnerability discovery rates. Zimmermann's [45] utilize logistic regression and SVMs to examine the predictability of software vulnerabilities in Windows Vista. The research reveals that while classical software metrics can predict vulnerabilities with reasonable precision, their recall rates are significantly low.

Russo [34], Gonzalez [14], and Okutan [29] utilize several machine learning classifiers for the automated characterization of software vulnerabilities based on CVE descriptions. Innovatively, Gonzalez [14] and Okutan [29] introduce a novel methodology using the NIST Vulnerability Description Ontology (VDO) framework. Okutan [29] goes further and employ Information-Theoretical methods for feature extraction. It is conducted through context-aware feature extraction and vectorization techniques, employing Term Frequency-Inverse Document Frequency (TF-IDF) and n-grams to account for the context of terms within CVE descriptions. Russo [34] employ a suite of algorithms including J48, BayesNet, NaiveBayes, Simple Logistic, and Random Forest, whereas Gonzalez [14] and Okutan [29] employ Naïve Bayes, Decision Tree, Support Vector Machine (SVM), AdaBoost-SVM, Random Forest, and a Majority Vote ensemble method combining predictions from the individual classifiers. The SVM and Decision Tree classifiers emerged as the most effective ones in accurately characterizing vulnerabilities based on textual descriptions from CVEs.

Parallel to these endeavors, Jimenez [19], Chen [8], and Madhushani [1] employ similar machine learning techniques to predict and classify software vulnerabilities. Jimenez [19] and Madhushani [1] utilize classifiers such as AdaBoost, J48, K-Nearest Neighbourhood, Logistic Regression, Random Forest, Naive Bayes, Stochastic Gradient Descent and Linear Support Vector (SVC). Madhushani [1] applies data mining techniques to predict future vulnerabilities' weaknesses based on the Bugs Framework from NIST. Results show the Linear SVC and Stochastic Gradient Descent models performing exceptionally well with accuracies up to 99% for precision, recall, and F1-score. Chen's [8] work on automating the curation of vulnerability databases underscores the potential of integrating complex machine learning models for enhanced vulnerability management. They word embedding and a stacking ensemble of classifiers with logistic regression as the meta learner.

Wijayasekara [40] and Chen [7] explore the use of text mining techniques and apply a combination of Naïve Bayes, Decision Tree, SVC and Random Forest. Wijayasekara [40] use text mining on bug reports from publicly available databases to identify Hidden Impact Bugs (HIBs), showing that up to 88% of HIBs were correctly identified. While Chen [7] distinguish invalid CVE reports, showcasing a promising performance with an AUC score over 0.8. The methodology used by both demonstrates the promising applicability of text mining in the early detection of vulnerabilities.

Collectively, these studies illuminate the diverse approaches and challenges in employing machine learning for the classification and prediction of software vulnerabilities. From the utilization of NVD data and advanced text mining techniques to the application of a wide range of machine learning models, the research not only showcases the potential of these methodologies in enhancing cybersecurity efforts but also highlights the need for further refinement and exploration of additional data sources and machine learning techniques to improve the accuracy and efficiency of vulnerability detection and classification.

3.2.2 Deep Learning

Central to this exploration is the deployment of deep learning models for the nuanced detection of vulnerabilities within code. The integration of traditional feature extraction methods with deep learning models, as exemplified by Huang's [18] TFI-DNN model, marks a notable innovation in the field. They focus on developing a deep learning model to automate the detection of software vulnerabilities from source code by the TFI-DNN model, which integrates TF-IDF, Information Gain (IG), and Deep Neural Networks (DNN). Feature extraction involves using TF-IDF for word frequency and weight calculation, and IG for optimal feature word selection, before applying the DNN for classification. The TFI-DNN model outperforms traditional machine learning models like SVM, Naive Bayes, and KNN across multiple dimensions—accuracy, recall, precision, and F1-score—showing significant improvements and validating the effectiveness of combining TF-IDF, IG, and DNN for vulnerability classification.

Similarly, Williams [41] aims to develop a comprehensive framework for analyzing and predicting

software vulnerabilities using data mining techniques. It applies feature extraction methods such as entity detection and TF-IDF for text analysis. The framework include the Topically Supervised Evolution Model (TSEM) for identifying temporal patterns in vulnerability data, diffusion-based storytelling for tracing the evolution of specific vulnerabilities, regression algorithms for predicting future vulnerabilities, and a deep neural network (DNN) for classifying vulnerabilities based on their features. The deep neural network (DNN) showcased high accuracy in classification tasks, and the regression models provided reliable predictions for future vulnerabilities.

Further enriching the landscape, Han [15] presents a deep learning approach to predict the multi-class severity level of software vulnerabilities based solely on their descriptions. The study employs word embeddings and a shallow Convolutional Neural Network (CNN) to automatically extract and utilize discriminative word and sentence features from vulnerability descriptions. The results demonstrate that their CNN model, trained on data from the CVE Details website, significantly outperforms baseline methods in predicting the severity levels of vulnerabilities as well as other deep learning architectures. Mazuera [27] critically evaluate various machine learning approaches, including deep and shallow learning techniques for software vulnerability detection. Their analysis spans Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Random Forest (RF) models, applied to both binary and multi-class classification tasks involving synthetic and real-world code vulnerabilities. The study finds that the Random Forest model outperforms deep learning models in terms of accuracy and F1 score, challenging the prevailing assumption of deep learning's superiority in this domain. This outcome emphasizes the importance of empirical testing to determine the most effective machine learning strategies for security applications.

Addressing the challenge of real-world applicability, Chakraborty [6] critically examine the performance of Deep Learning (DL) techniques in real-world software vulnerability prediction, identifying a significant accuracy decline when models proven to be up to 95% accurate on curated datasets are applied to real-world data, with a noted accuracy reduction of over 50%. Addressing this gap, their study contrasts various models, including token-based approaches like BLSTM and sophisticated graph-based techniques such as the Devign model, which utilizes Code Property Graphs (CPG) for deeper semantic and syntactic analysis. The research significantly improves upon traditional DL methodologies in precision and recall by incorporating real-world project data from sources like Chromium and Debian and advancing towards more semantically aware models. This shift not only highlights the necessity for realistic data collection and advanced model development but also marks a considerable advancement in DL's application to cybersecurity, challenging and surpassing existing models with marked enhancements in model performance.

Futhermore, Steenhoek [36] inspects the capability of deep learning models in identifying software vulnerabilities, using the Devign and MSR datasets enriched with pre-existing annotations for software vulnerabilities. These datasets facilitate a comprehensive comparison across a spectrum of machine learning models, such as Graph Neural Networks (GNNs), Recurrent Neural Networks

(RNNs), Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), and Transformers. This comparison aims to determine their efficacy in pinpointing various vulnerability types. The study meticulously examines the impact of dataset size and composition on the performance of these models, revealing significant variability in their effectiveness. Certain models excel in specific areas but do not maintain a consistent performance across all vulnerability types. The comparative results shed light on the distinct advantages and limitations of each model in the context of software vulnerability detection.

Zhen [24], [22], [23] series of papers focus on improving the precision of software vulnerability detection with deep learning-based models using source code from C/C++ programs. Zhen [24] present VulDeePecker, a deep learning system using Bidirectional Long Short-Term Memory (BLSTM) neural network and focusing specifically on buffer errors (CWE-119) and resource management errors (CWE-399). VulDeePecker's performance significantly surpasses that of traditional static analysis tools and code similarity-based systems in terms of reducing false negatives while maintaining reasonable false positives. While, Zhen [22] employ Bidirectional Gated Recurrent Unit (BGRU) model, RNNs, Convolutional Neural Networks (CNNs), Deep Belief Networks (DBNs), and shallow learning models. The BGRU model demonstrated superior performance in identifying vulnerabilities, highlighted by the detection of 15 previously unreported vulnerabilities in four software products. Zhen [23] leverages a novel dataset consisting of 157,692 vulnerability candidates in LLVM intermediate code, of which 40,450 are labeled as vulnerable. It uses the VulDeeLocator which shows an average improvement of 9.8% in F1-measure, 7.9% in false-positive rate, and 8.2% in false-negative rate over state-of-the-art models. Additionally, it demonstrates a 4.2 times increase in locating precision when tested on real-world software products. This suggests a substantial step forward in the application of deep learning to software vulnerability detection and location.

3.2.3 Transformers

On another front, Fu [12], Chandra [37], and Gao [13] use more complex methods for the detection of software vulnerabilities, such as transform-based models. Chandra [37] conduct a comprehensive study on the efficacy of machine learning models for detecting vulnerabilities in C/C++ codebases, employing two datasets VulDeePecker and SeVC. The study compared the performance of several models, including traditional RNNs like BiLSTM and BiGRU, alongside advanced transformer-based models such as BERT, GPT-2, CodeBERT, and DistilBERT. The analysis revealed that transformer-based models, with a particular emphasis on GPT-2 Large and XL versions, significantly outperformed RNN models in detecting software vulnerabilities, achieving higher scores in precision, recall, and F1 metrics.

On a similar note, Omar [30] aims to enhance software vulnerability detection by introducing a novel knowledge distillation (KD) technique. It utilizes public domain datasets, specifically SARD, SeVC, Devign, and D2A, and employ GPT-2, CodeBERT, and LSTM, with a focus on

the application of the KD method. The results revealed that DistilVulBERT, a product of the KD process, significantly outperformed the other models, achieving a remarkable score of 94.0% on these datasets. This score underscores the effectiveness of integrating transformer-based models with knowledge distillation for improved accuracy in identifying software vulnerabilities.

Fu [12] developed LineVul, a groundbreaking method for predicting software vulnerabilities at the line level within C/C++ code using the Transformer architecture. Their work use CodeBERT, Random Forest, IVDetect, Reveal, SySeRV, VulDeePecker, Devign. LineVul surpasses existing models like IVDetect, Reveal, and VulDeePecker in accuracy and efficiency, achieving up to 379% improvement in F1-measure for function-level predictions and 25% higher Top-10 Accuracy for line-level predictions, setting a new benchmark for vulnerability detection methods. This approach leverages the Big-Vul dataset, an extensive collection of over 188,000 C/C++ functions from 348 open-source GitHub projects, encompassing 91 Common Weakness Enumerations (CWEs) and data from 2002 to 2019. The dataset was compiled through a detailed examination of the Common Vulnerabilities and Exposures (CVE) database and GitHub repositories, resulting in a repository of 3,754 code vulnerabilities.

Utilizing various Large Language Models, Gao [13] introduce VulBench, a benchmark designed to assess the efficacy of Large Language Models (LLMs) in detecting software vulnerabilities. It utilizes human-labeled data from diverse sources such as Capture-the-Flag challenges and datasets like MAGMA, Devign, D2A, and Big-Vul. The study conducts a thorough analysis comparing 16 LLMs, including GPT-3.5, GPT-4, and various Llama2 models, against traditional deep learning models and static analyzers. GPT-4 emerges as the top performer, demonstrating superior capability in binary and multi-class classification tasks with impressive F1 scores and precision. Those researches highlights the remarkable potential of LLMs in advancing software security through effective automated vulnerability detection.

Author	Goal	Databases	Models
Chen [8]	Automate curation of software vulnerability databases	Jira tickets, Bugzilla reports, CVEs, emails, GitHub issues and commits.	Random Forest, Naive Bayes, KNN, Support Vector Machine, Gradient Boosting, AdaBoost, Logistic Regression
Okutan [29]	Automated curation/characterization of software vulnerabilities	NVD	SVM, Naïve Bayes, C4.5 Decision Tree, Random Forest, Ensemble Learning
Gonzalez [14]	Automated characterization of software vulnerabilities based on CVE descriptions	NVD	Naïve Bayes, Decision Tree, Support Vector Machine, AdaBoost-SVM, Random Forest, Majority Vote
Chen [7]	Identifying and predicting invalid CVE reports	NVD	Naive Bayes, Multinomial Naive Bayes, SVM, Random Forest
Russo [34]	Developing a tool designed to summarize and categorize CVEs	NVD	J48, BayesNet, Naive Bayes, Simple Logistic, RandomForest
Nguyen [26]	Evaluating data sources for software vulnerabilities in Mozilla Firefox	NVD, MFSA, Bugzilla	
Jimenez [19]	Examination of the effectiveness of vulnerability prediction models under realistic conditions	Linux Kernel, OpenSSL, Wireshark	AdaBoost, J48, KNN, Logistic Regression, Random Forest
Williams [41]	Analyzing and predicting software vulnerabilities using data mining techniques	NVD	TSEM, Linear Regression, Exponential-Weighted Least-Squares Regression, Logarithmic-Weighted Least-Squares Regression, Cauchy Regression, DNN
Zimmermann [45]	Examiation of the predictability of software vulnerabilities in Windows Vista	NVD	Logistic Regression, Support Vector Machine
Last [21]	Predicting future software vulnerabilities	NVD	Linear Regression, Quadratic Regression, KNN
Zhang [43]	Predicting the time to the next software vulnerability for software applications	NVD	Linear Regression, Least Median Square, Multilayer Perceptron, Radial Basis Function (RBF) Network, Sequential Minimal Optimization (SMO) Regression, and Gaussian Processes.
Han [15]	Predicting the multi-class severity level of software vulnerabilities	CVE	Shallow CNN
Huang [18]	Automating the detection of software vulnerabilities from source code	NVD	TFI-DNN, Support Vector Machine, Naive Bayes, KNN
Madhushani [1]	Classifying software vulnerabilities based on Bugs Framework	NVD, CVE	Linear SVC, Random Forest, Multinomial Naive Bayes, Logistic Regression, Naive Bayes, Stochastic Gradient Descent
Wijayasekara [40]	Identifying Hidden Impact Bugs in software using text mining	Redhat Bugzilla, CVE	Naive Bayes, Naive Bayes Multinomial, Decision Tree
Fu [12]	Fine-grained, line-level software vulnerability prediction using Transformer architecture	Big-Vul	CodeBERT, Random Forest, IVDetect, Reveal, SySeRV, VulDeePecker, Devign
Gao [13]	Evaluating Large Language Models in vulnerability detection	Capture-the-Flag, MAGMA, Devign, D2A, Big-Vul	GPT, Llama-2, Vicuna, Baichuan, Internlm, CodeLlama, Platypus2, Falcon-40, ChatGLM2
Zhou [44]	Identifying vulnerabilities in software using a graph neural network model	Open-sourced C projects	Graph Neural Network
Mazuera [27]	Testing efficacy of machine learning models for detecting software vulnerabilities	SARD, NVD, GitHub Archive Dataset (GH-DS).	CNNs, RNNs, Random Forest
Chakraborty [6]	Predicting software vulnerabilities under real-world conditions	Chromium, Debian	BLSTM, Devign, CNN, RNN, VulDeePecker, SySeVR
Chandra [37]	Identifying vulnerabilities in C/C++ codebases using machine learning models	VulDeePecker, SeVC	RNNs (BiLSTM, BiGRU), Transformer-based models (BERT, GPT-2, CodeBERT, DistilBERT)
Omar [30]	Enhancing software vulnerability detection with knowledge distillation	SARD, SeVC, Devign, and D2A	GPT-2, CodeBERT, LSTM
Li [24]	Detecting software vulnerabilities focusing on buffer errors and resource management errors	NVD, SARD	BLSTM
Li [22]	Detecting vulnerabilities in C/C++ programs using deep learning	NVD, SARD	Bidirectional RNNs (BGRU), RNN, CNNs, DBNs
Li [23]	Improving precision of software vulnerability detection through a deep learning model	LLVM	VulDeeLocator
Steenhoek [36]	Evaluate the effectiveness of various deep learning models for vulnerability detection within software code	Devign, MSR	Devign, ReVeal, ReGVD, Code2Vec, CodeBERT, VulBERTa-CNN, VulBERTa-MLP, PLBART, and LineVul

Table 3.1: A survey of methods for the curation, prediction, and classification of software vulnerabilities.

CHAPTER 4
METHODOLOGY

Our research introduces an approach that leverages the power of transformers to train classifiers capable of analyzing Common Vulnerabilities and Exposures (CVE) reports and inferring their associated Vulnerability Description Ontology (VDO) characteristics. We hypothesize that the lexicon utilized within CVE descriptions contains valuable indicators of these characteristics, and through machine learning, classifiers can be trained to recognize these indicators from historical data. Fig. 4.1 provides a general overview of our proof of concept for examining the applicability of the proposed methodology.

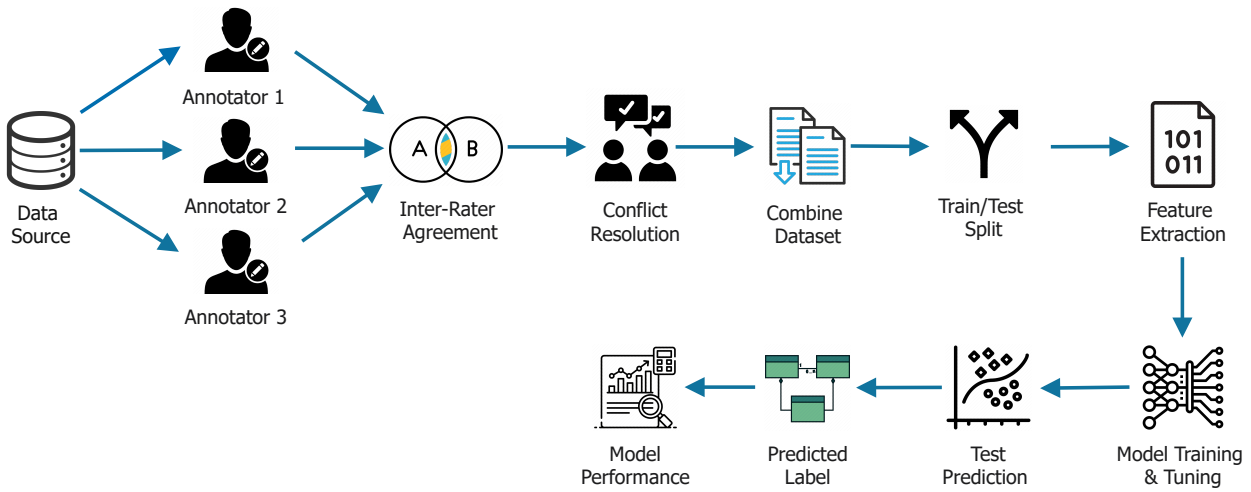

Figure 4.1: An illustration of the proof of concept that is designed to examine the applicability of CVE descriptions classification.

In summary, we engage in a multi-stage process to validate our approach:

1. We create a labeled dataset through the manual curation of vulnerability descriptions corresponding to each of the 27 VDO characteristics. In addition to that, we reuse the dataset from Okutan [29] and merge it with ours.

2. Transform-based classifiers are trained using this dataset. The primary goal is to predict the one or more classes for text descriptions within each of the five distinct noun groups. These groups feature different classification schemes:

    - Multi-class classification is used when a text description corresponds to a single label, suitable for categories like Attack Theater, Context, and Impact Method noun groups. In these groups, the classes are mutually exclusive.

- Multi-label classification applies to scenarios where a text description fits multiple categories, such as in Logical Impact and Mitigation noun groups. These descriptions may fall into one or more classes. For instance, in the Logical Impact category, an attacker with access to a user's data could both read and write to it.

3. Finally, we compute a set of evaluation metrics to benchmark the classifiers' performance.

4.1 Data Source

Research in software vulnerability detection employs a variety of datasets to enhance security through machine learning, with the comprehensive National Vulnerability Database (NVD) being a focal point for many studies. This database supports a wide range of research objectives, including automated vulnerability classification, future vulnerability prediction, and severity score prediction. In particular, the NVD has underpinned the work of numerous researchers such as Zhang [43], Zimmermann [45], Last [21], and others, showcasing its extensive utility across the field.

In addition to the NVD, other critical datasets include the Software Assurance Reference Dataset (SARD), and specialized collections like Big-Vul or LLVM intermediate code dataset. These datasets facilitate various research avenues, from severity prediction using CVE as seen in Han [15], to exploring vulnerabilities in C/C++ programs with SARD and Big-Vul by Fu [12], and automating database curation with a mix of Jira tickets, Bugzilla reports, and GitHub issues by Chen [8]. Furthermore, Chakraborty [6] focus on real-world conditions using data from Chromium and Debian, illustrating the diversity and significance of employing project-specific and proprietary data in advancing the field of software vulnerability detection.

4.1.1 Data Retrieval

For the purpose of our research, we employ the National Vulnerability Database (NVD) as our primary source of experimental data. The dataset represents the most complete and suitable dataset compared to other available options. The dataset from the NVD is organized as a series of JSON files, each file enumerating various attributes of the vulnerabilities such as the CVE identifier, the release date, the CVSS version and score, the CVSS vector, and a descriptive text of the vulnerability. The process involve querying the NVD Vulnerability Search Engine and extract the CVE descriptions information from the years 2021 to 2023 utilizing Python scripts.

4.2 Data Annotation

The annotation phase marks a critical moment in our research, requiring both a high degree of accuracy and an extensive understanding of cybersecurity frameworks. We strictly follow the rigorous curation standards that Okutan [29] previously established throughout our dataset development.

Our approach features a focus on achieving uniformity across the dataset, which is accomplished by adhering to a series of carefully devised steps by three student annotators:

4.2.1 Annotation Sessions

1. *Understanding the Vulnerability Description Ontology (VDO)*: The first step requires annotators to engage in a detailed review of the VDO framework and the dataset annotated by five security specialists with extensive experience, as reported by Okutan [29].

2. *Achieving Unanimity in Class Definitions*: The second step requires annotators to participate in discussions to ensure a unified understanding of the classes associated with each noun group.

3. *Dataset Construction and Annotation*: The step requires constructing the datasets for the five distinct VDO noun groups. The annotators receive identical CVE descriptions and independently annotate them within each noun group. The CVE data points must include labels along with a confidence score on a numerical scale ranging from one to three, where a score of one indicated uncertainty, two denoted a moderate level of confidence, and three reflected a high level of certainty in the labeling decision. This scoring system was designed to discern high-confidence annotations from annotations that require a peer-review to reduce speculative labeling.

CVE - ID	**Description**	**HPKP/HSTS**	**MFA**
CVE-2019-6531	An attacker could retrieve passwords from a HTTP GET request from the Kunbus PR100088 Modbus gateway versions prior to Release R02 (or Software Version 1113166) if the attacker is in an MITM position.	x	x
CVE-2019-18666	An issue was discovered on D-Link DAP-1360 revision F devices Remote attackers can start a telnet service without authorization via an undocumented HTTP request. Although this is the primary vulnerability the impact depends on the firmware version. Versions 609EU through 613EUbeta were tested, versions through 612b01 have weak root credentials allowing an attacker to gain remote root access. After 612b01 the root credentials were changed but the telnet service can still be started without authorization.	x	

Table 4.1: CVEs labeled for two classes in the Mitigation noun group, i.e., HPKP/HSTS (HTTP Public Key Pinning and HTTP Strict Transport Security) and Multi-Factor Authentication (MFA).

The complexity of certain CVEs necessitate the assignment of multiple classes within a noun group to accurately capture their multifaceted nature. In such cases, annotators were instructed to apply all pertinent labels, ensuring comprehensive coverage of the CVE's characteristics and precision in our dataset. These multi-label instances are documented and exemplified in Table 4.1.

4.2.2 Inter-Rater Agreement

Inter-rater reliability is a critical component of data annotation, providing a measure of consistency between independent annotators. Our approach is based on the principle that data become reliable when annotators agree on how to categorize units of analysis. This principle is a well-established concept in content analysis research, as noted by Krippendorff [20]. It underscores the importance of inter-rater agreement for achieving trustworthy data.

To quantitatively assess the degree of agreement between annotators, we implemented the MASI (Metric for Agreement with Set Intersection) score, which is a metric designed to measure the agreement between two sets by evaluating the overlap of assigned labels. This metric is particularly suited for annotations where multiple labels may be applicable to a single instance. MASI's flexibility allows it to remain agnostic to the probability distribution of annotations, enabling its use alongside various weighted agreement metrics, such as the widely-recognized Krippendorff's α Passonneau [32]. The MASI scores for each noun group were calculated and are presented in Table 4.2. These scores provide a statistical basis for assessing the consistency of annotations across the three annotators.

$$\alpha = 1 - \frac{D_o \text{ observed disagreement}}{D_e \text{ expected disagreement}} \tag{4.1}$$

Furthermore, Krippendorff's α, as discussed in Carletta [5] offers a scale of agreement where a score of zero indicates no agreement beyond chance, and a score of one represents complete agreement. Specifically, Krippendorff outlines thresholds for acceptable levels of agreement based on the intended use of the coding. For instance, it is generally challenging to establish correlations between two variables if both are based on coding schemes where α is less than 0.7. Moreover, content analysis researchers consider α values above 0.8 as indicative of strong reliability, whereas values between 0.67 and 0.8 permit only tentative conclusions.

Our initial analysis across various noun groups yielded the following preliminary scores: Attack Theater at 0.68, Context at 0.66, Mitigation at 0.76, Logical Impact at 0.77, and Impact Method at 0.74, with an overall average of 0.72. These preliminary results indicate that while some categories are nearing the threshold for reliable conclusions, others remain in the range where only tentative conclusions can be drawn. Nevertheless, disagreements among annotators are subsequently addressed to resolve discrepancies, aiming to enhance the reliability of the annotations.

Noun Group	Alpha Rate (α)
Attack Theater	0.68
Context	0.66
Mitigation	0.76
Logical Impact	0.77
Impact Method	0.74
AVERAGE	***0.72***

Table 4.2: MASI scores for each noun group by three annotators.

4.2.3 Resolution of Annotation Discrepancies

A rigorous review process can significantly improve the data's reliability and validity, which is crucial for developing computational models. Post-annotation, all CVEs and associated confidence scores were subjected to a peer review.

1. *Confidence Score*: Annotators flagged CVEs for further discussion and refined the dataset until all entries achieved a confidence score of three.

2. *Conflict Resolution*: If there were disagreements between annotators in labeling a CVE, its description and its characteristics were discussed in an attempt to resolve the conflict. If the annotators were not able to reach a consensus, the CVE description was remove from the dataset.

As a result, the newly annotated dataset reflects a consensus among annotators, providing a scientifically valid foundation for our research into vulnerability classification. This step produced our final dataset, which was then combined with the dataset from [29]. The detailed counts for each class within each noun group are presented in Table 4.3. The table outlines the counts from the dataset reused from [29], and the counts from the new dataset annotated by our team.

4.3 Model Development

4.3.1 Train/Test Split

The evaluation of the model's effectiveness necessitates subjecting it to examination against a held-out test set containing CVE descriptions with pre-determined classes. This standard approach is essential for ensuring the accuracy and reliability of the model's performance within the domain of machine learning research. The dataset is divided into 80% for training and 20% for testing. The stratified sampling techniques guarantees a well-balanced representation across diverse classes, minimizing potential biases during evaluation process.

Noun Group	Class	Okutan et al.	New Dataset	Total Count
Attack Theater	Remote	91	80	171
	Local	75	87	162
	Limited Remote	74	67	141
	Physical	53	101	154
Context	Application	156	75	231
	Hypervisor	102	34	136
	Firmware	103	69	172
	Physical Hardware	129	59	188
	Channel	103	73	176
	Host OS	102	53	155
	Guest OS	103	58	161
Impact Method	Trust Failure	86	127	213
	Context Escape	68	127	195
	Authentication Bypass	111	110	221
	Man-in-the-Middle	79	107	186
	Code Execution	121	119	240
Logical Impact	Service Interrupt	77	89	166
	Read	116	107	223
	Write	134	108	242
	Privilege Escalation	70	92	162
	Resource Removal	95	111	206
	Indirect Disclosure	70	111	181
Mitigation	ASLR	96	95	191
	MFA	99	91	190
	Sandboxed	88	92	180
	HPKP/HSTS	92	104	196
	Physical Security	99	87	186

Table 4.3: Original research versus current study annotations.

4.3.2 Preprocessing

In the preprocessing phase for this study, the initial step involves removing URLs from text descriptions. This action is specific to the dataset, as URLs in the descriptions do not contribute to the predictive modeling task. The presence of URLs could potentially introduce noise, given their irregular structure and lack of relevant linguistic content for classification purposes.

Following the removal of URLs, the text undergoes tokenization, a process where it is dissected into tokens—units representing words or subwords. This transformation shifts unstructured text into a structured format, significantly enhancing the models' ability to process and interpret the data. Each model employs a specific tokenizer optimized for its architecture: BertTokenizer for

BERT, DistilBertTokenizer for DistilBERT, and AutoTokenizer for XLNet and DeBERTa, each designed to accommodate their respective architectural variations. For example, BertTokenizer utilizes the WordPiece algorithm to segment text into a mix of whole words and recognizable subwords, efficiently addressing the challenge of a fixed vocabulary size [11]. In contrast, AutoTokenizer dynamically adapts its tokenization strategy to suit each model's unique contextual embedding methodologies.

Particularly, the preprocessing requirements for large language models like those selected for this study, are inherently less demanding due to their extensive pre-training on diverse corpora. This pre-training enables LLMs to effectively handle a variety of textual inputs with minimal preprocessing, distinguishing them from models that might rely more heavily on specific preprocessing steps to achieve optimal performance. Despite this, the decision to remove URLs prior to tokenization is maintained across all models to ensure uniformity in the text input, aligning with the overall aim of minimizing non-linguistic data that could reduce the models' learning ability.

4.3.3 Fine-tuning

In this research work we use 5 transform-based models as illustrated in Fig. 4.2:

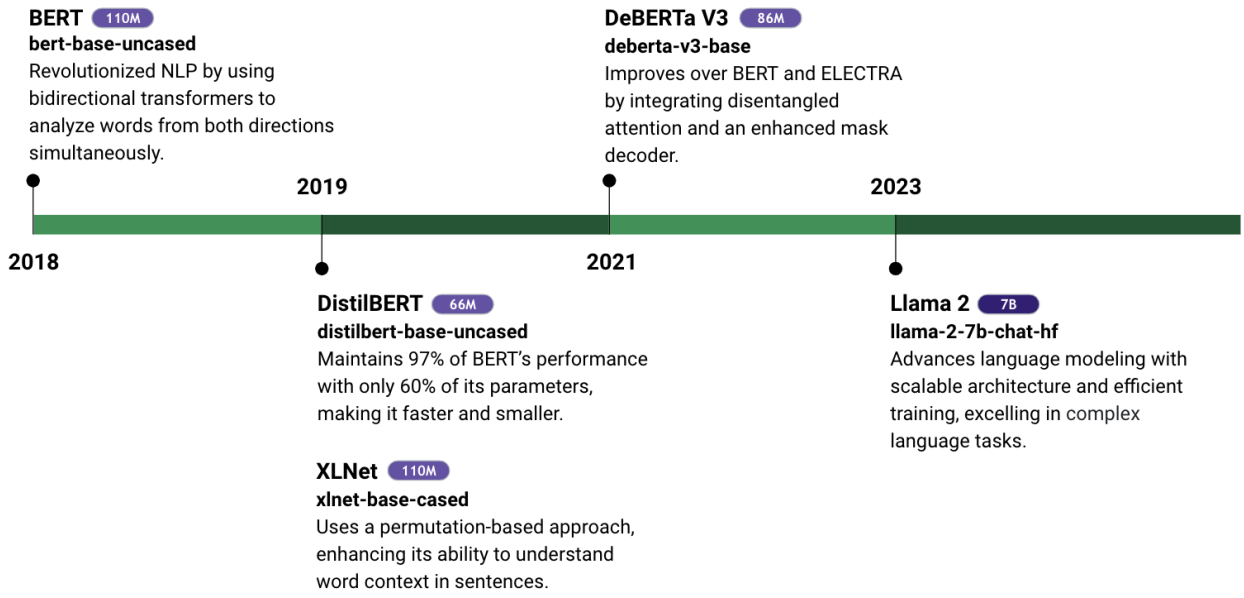

Figure 4.2: Number of parameters and development years of models employed in the research.

In 2018, BERT revolutionized transformer models by introducing bidirectional context analysis, a significant advancement over previous transformers (Devlin [11]). This breakthrough set the stage for subsequent innovations. In 2019, DistilBERT emerged with a focus on efficiency, maintaining 97% of BERT's performance while reducing parameter size to 66 million, compared to BERT's 110 million (Sanh [35]). Also in 2019, XLNet adopted a permutation-based approach, enhancing its

contextual understanding (Yang [42]). In 2021, DeBERTa V3 integrated disentangled attention and an enhanced mask decoder, surpassing BERT and ELECTRA in performance. Finally, in 2023, Llama 2 introduced a scalable architecture with billions of parameters, achieving excellence in tasks like text summarization (Touvron [38]).

These models are fine-tuned on the dataset, employing sequence classification heads: XLNetForSequenceClassification, DistilBertForSequenceClassification, BertForSequenceClassification, and DebertaForSequenceClassification. These classification heads are specialized layers attached to the output of the transformer models, designed to interpret the representations produced by the transformers' encoders for the task at hand.

- *Multi-class Classification*: The CrossEntropyLoss function is employed for tasks requiring the prediction of a single class in N classes, where N is bigger than 2. This loss function is particularly effective in scenarios where only one class is correct, encouraging the model to increase the probability of the true class while decreasing that of all others. Specifically, CrossEntropyLoss quantifies the difference between the predicted probability distribution across all classes and the actual distribution, where the true class has a probability of 1 and all others 0. It utilizes a softmax function to convert the models' logits into probabilities by comparing the logits of all possible classes. Logits are raw non-normalized predictions generated by the last layer of the network. This process effectively assigns each text description to a single, most probable label Fig. 4.3.

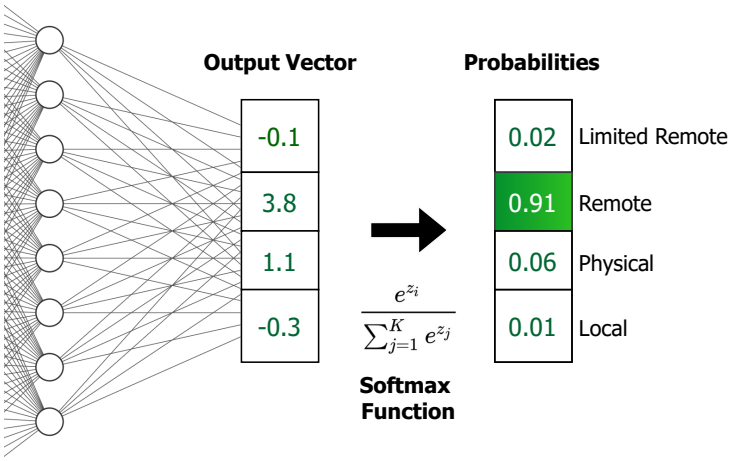

Figure 4.3: Multi-class Softmax function using Attack Theater noun group as an example.

- *Multi-label Classification*: Conversely, in multi-label classification scenarios, where text descriptions may concurrently belong to multiple categories, BCEWithLogitsLoss (Binary Cross Entropy with Logits Loss) is utilized. This function computes the binary cross-entropy loss between the target and the input logits, effectively handling multiple labels by treating each

label as an independent Bernoulli distribution. These heads apply a sigmoid function to each output logit independently. This approach allows for the assignment of probabilities to each label independently, enabling the model to predict multiple labels for a single text instance by considering each label as a separate binary classification problem Fig. 4.4.

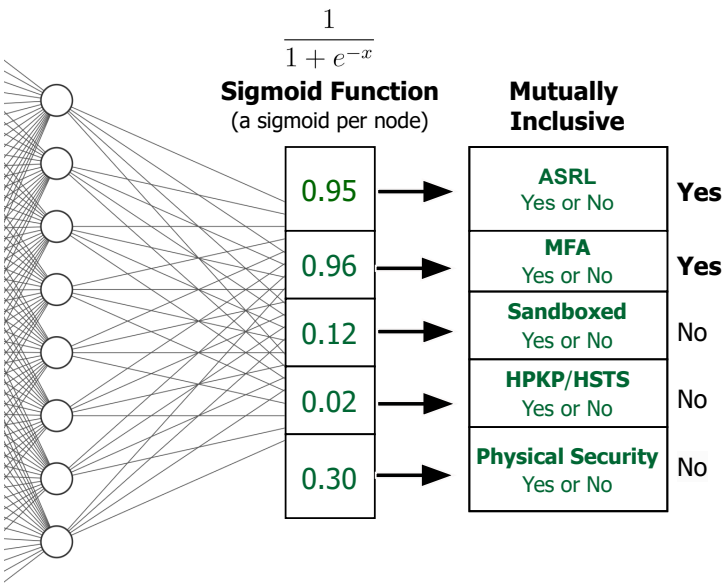

Figure 4.4: Multi-label Sigmoid function using Mitigation noun group as an example.

Llama 2 Prompt

The configuration of prompts was designed to support the model's task of analyzing and classifying software vulnerability descriptions. By specifying structured prompts, the model is trained to accurately classify textual data into predefined categories such as 'Authentication Bypass,' 'Code Execution,' and 'MitM.' These prompts are integral to guiding the model through the task of categorizing different types of vulnerabilities. An example of such a prompt is as follows:

Example Prompt

Analyze the software vulnerability description enclosed in square brackets, and categorize it into one of the Impact Method classes: Authentication Bypass, Code Execution, Context Escape, MitM, Trust Failure. Return the answer as the corresponding class label.
[Jenkins SmallTest Plugin 1.0.4 and earlier does not perform hostname validation when connecting to the configured View26 server that could be abused using a man-in-the-middle attack to intercept these connections.] = MitM.

Hyperparameter Tuning for BERT, DistilBERT, XLNet, DeBERTa V3

To optimize the fine-tuning of transformer-based models for label prediction tasks, a systematic approach to hyperparameter tuning was undertaken, informed by machine learning principles and empirical testing.

- *Epochs*: The models were trained for up to 30 epochs, with an early stopping mechanism that uses a patience of three epochs based on test loss. This method is informed by the observation that the test loss often plateaus well before reaching 30 epochs. As illustrated in Fig. 4.5 and Fig. 4.6, early stopping is triggered when the test loss does not decrease for three consecutive epochs, effectively preventing overtraining by halting further training when additional epochs do not yield improvement in test loss. This approach ensures efficient training by saving computational resources and potentially enhancing model generalizability by avoiding overfitting to the training data.

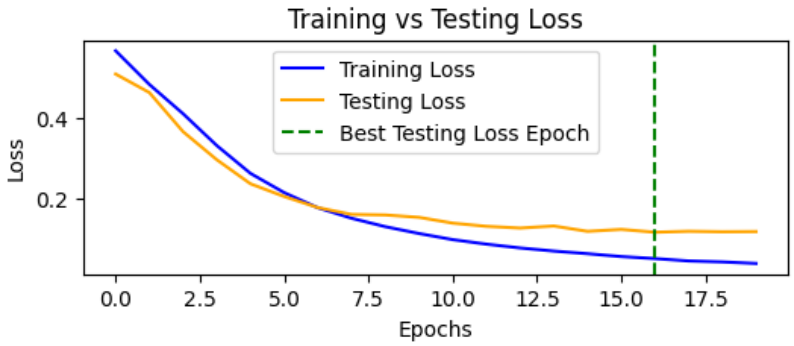

Figure 4.5: BERT model loss plot for the Logical Impact category using the combined dataset.

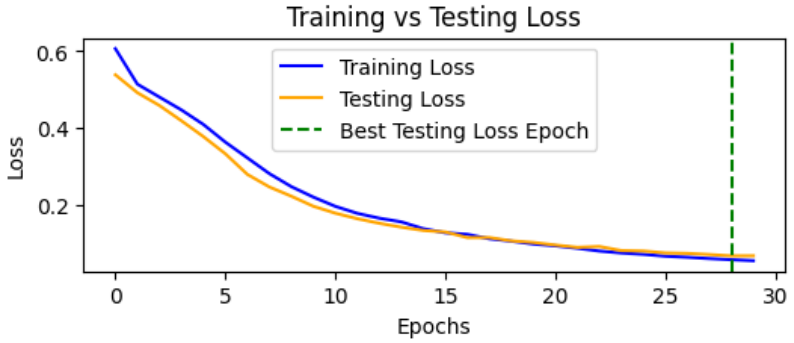

Figure 4.6: BERT model loss plot for the Logical Impact category using only Okutan's dataset.

- *Optimizer*: The Adam optimizer was selected for the optimization process since it is able to navigate high-dimensional parameter spaces characteristic of transformer models, crucial for learning complex patterns within the data.

- *Learning Rate*: Initial tests with a learning rate of 1e-4 revealed unstable learning dynamics, as evidenced by sharp fluctuations in the loss function. A subsequent reduction to 2e-5 marginally improved stability but did not achieve optimal results. The final adjustment to a learning rate of 1e-5 was determined based on its superior balance between learning efficiency and stability, as lower learning rates generally promote gradual but consistent convergence in deep learning models.

- *Weight Decay*: The weight decay parameter was set to 1e-5, complementing the chosen learning rate to foster stable learning progress. Alternative decay rates, including 0.1, 0.01, 0.001, and 0.0001, were evaluated but resulted in inferior learning behavior. The selected decay rate of 1e-5 is supported by its effectiveness in regularizing the model, mitigating the risk of overfitting by penalizing larger weights. This choice is substantiated by the principle that appropriate weight decay can enhance generalization in deep neural networks by encouraging simpler models that perform well on unseen data.

- *Maximum Sequence Length*: Experimental trials with sequence lengths of 128 and 200 offered no significant benefit over the baseline setting of 256. This parameter was carefully chosen considering the typical length of text descriptions in the dataset, which rarely exceeded 300 words. A sequence length of 256 balances between encompassing the entirety of most descriptions and maintaining computational efficiency. It acknowledges the practical observation that while most descriptions are shorter than 110 words, a length of 256 accommodates the full range of observed text lengths without unnecessary padding.

The fine-tuning configurations are consistent across all models for comparability, except for the Llama2 model, which requires a unique setup detailed in Table 4.5. The optimal hyperparameters for each model are shown in Table 4.4.

Hyperparameters	**Values**
Epochs	30
Learning Rate	1e-5
Weight Decay	1e-5
Training Batch Size	16
Testing Batch Size	8
Maximum Sequence Length	256
Early Stopping Patience	3

Table 4.4: Optimal hyperparameters for BERT, DistilBERT, XLNet, Deberta V3.

Hyperparameter Tuning for LLaMA 2

- *Epochs*: Adjusting the number of training epochs was critical in avoiding overfitting while ensuring sufficient learning. While 3 epochs led to overfitting, 1 epoch was inadequate for the model to learn effectively. A balanced approach was achieved with 2 epochs, facilitating adequate exposure to the training data without the adverse effects of overtraining.

- *QLoRA Parameters*: The configuration of QLoRA parameters started with a dimensionality rank of 64 and a scaling factor alpha of 16, selected to optimize the model's pattern recognition capacity while maintaining computational efficiency. Trials involving variations of rank and alpha (testing rank: 64 with alpha: 32, and the inverse) revealed that the original configuration offered the best results.

- *Learning Rate and Weight Decay*: The initial rates of 1e-5 and 5e-5 led to underfitting, and a higher rate of 1e-4 did not achieve the expected performance. A learning rate of 2e-4, coupled with the Adam optimizer, was identified as the best value. Concurrently, a weight decay of 0.001 was chosen to minimize overfitting by penalizing larger weights, which helps maintain steady learning progress. This approach proved more effective than a previously tested decay rate of 0.01, which led to less favorable outcomes.

- *Batch Size and Gradient Accumulation*: The model's training regimen included a per-device training batch size of 1 and a gradient accumulation strategy spanning 8 steps. This approach was designed to optimize computational resources, allowing for effective model updates and handling of larger sequences without surpassing memory limits, a key consideration for models with extensive parameter counts.

- *Advanced Configurations*: Implementations such as gradient checkpointing and mixed-precision training (fp16) were strategic choices aimed at enhancing memory utilization and computational speed, essential for efficiently managing a model of this scale and complexity.

Hyperparameters	**Values**
Alpha	16
Rank	64
Epochs	2
Learning Rate	2e-4
Weight Decay	1e-3
Maximum Sequence Length	512
Training Batch Size	1
Gradient Accumulation	8

Table 4.5: Optimal hyperparameters Llama 2.

4.4 Evaluation Metrics

To assess the performance of our classification model and facilitate comparisons with other algorithms, it's essential to use a unified set of evaluation criteria. In this study, we have chosen precision, recall, and the F1-score, which are widely accepted metrics for evaluating the effectiveness of classification models, as referenced in multiple studies [4], [6], [10].

To apply the binary classification metrics of precision, recall, and the F1-score to our multi-class scenario, we calculate these metrics for each class as if it were a binary classification. We then average the scores across all classes to derive a comprehensive measure of the model's performance. Before introducing the evaluation metrics utilized in this study, we establish the definitions of several key terms:

- **True Positive (TP)** refers to instances where the model correctly predicts a sample as belonging to a specific class.
- **False Positive (FP)** denotes instances where the model incorrectly predicts a sample as belonging to a target class when it belongs to a different class.
- **True Negative (TN)** represents instances where the model correctly identifies a sample as not belonging to the target class.
- **False Negative (FN)** occurs when the model fails to identify a sample as belonging to the target class, classifying it under a different class instead.

Recall quantifies the model's ability to correctly identify positive examples for a given class i. It is the proportion of true positives in relation to the sum of true positives and false negatives (FN), representing the model's capacity to capture all relevant instances. The calculation formula is below.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.2}$$

Precision evaluates the correctness of positive predictions for a given class i. It is the ratio of true positives to the sum of true positives and false positives, reflecting the model's precision in classifying instances as positive. The calculation formula is below.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.3}$$

F1-score is the harmonic mean of precision and recall. For a given class i, the F1-score combines these two metrics, providing a balance between precision and recall in a single measure. This is particularly useful for comparing model performance across different class distributions. The F1-score ranges between 0 and 1, where 1 indicates perfect precision and recall, and hence, perfect

classification performance for the given class. We employ the F1-score as the main evaluation metric due to its balanced consideration of both precision and recall. The calculation formula is below.

$$\text{F1-score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \tag{4.4}$$

CHAPTER 5
RESULTS

5.1 RQ1: Does the expansion of the dataset lead to enhanced performance in vulnerability classification by large language models?

The first question addresses whether extending the data can lead to stronger models capable of categorizing a wider array of vulnerability types. This analysis seeks to determine the benefits of extending the dataset by comparing F1 scores from models trained on both the original Okutan dataset [29], results in Table 5.2 and a combined dataset with new annotations, results in Table 5.1. This examination is key to understanding how the size of data influences the accuracy and generalization ability of LLMs in categorizing software vulnerabilities.

Attack Theater

For Limited Remote vulnerabilities, the combined dataset shows a range of accuracies from 0.60 (BERT) to 0.79 (Llama 2), which, when compared to the results using only Ahmet et al.'s data, indicates a notable decrease. Specifically, the original dataset alone resulted in higher accuracies for Llama 2 at 0.89 and DeBERTa V3 at 0.87. In the Local attribute, the combined dataset's accuracies vary from 0.69 (Llama 2) to 0.81 (XLNet and DeBERTa V3), contrasting with the original dataset's higher accuracies, such as 0.91 for DistilBERT and 0.89 for BERT. This indicates that the expanded dataset does not necessarily translate to improved performance and, in some cases, may slightly hinder it.

The Remote attribute's comparison further illustrates this trend, where the combined dataset yields accuracies ranging from 0.76 (Llama 2) to 0.87 (DistilBERT). However, the original dataset alone demonstrates superior performance, with accuracies reaching up to 0.95 for DistilBERT and 0.95 for DeBERTa V3. Lastly, the Physical vulnerabilities attribute presents a mixed picture. The combined dataset shows high accuracies, particularly with DistilBERT and Llama 2 achieving perfect scores of 1.00. Conversely, the original dataset's results are more varied, with Llama 2 still achieving a perfect score but other models like XLNet dropping to 0.44, indicating that in some cases, the increased dataset size can significantly enhance model f1 score by providing a richer set of examples for model training.

Context

For Application vulnerabilities, the expanded dataset results in Llama 2 achieving a significantly higher f1 score of 0.98, compared to 0.85 using only the original data. In the Channel attribute, the

Table 5.1: F1-score values for each attribute using combined dataset.

Attributes	**BERT**	**DistilBERT**	**XLNet**	**DeBERTa**	**Llama 2**	**AVG**
Limited Remote	0.60	0.63	0.62	0.71	**0.79**	0.67
Local	0.78	0.77	**0.81**	**0.81**	0.69	0.77
Remote	0.83	**0.87**	0.86	0.86	0.76	0.83
Physical	0.98	**1.00**	0.95	0.98	0.97	0.98
Application	0.91	0.90	0.91	0.92	**0.98**	0.92
Channel	0.86	0.85	**0.87**	0.83	0.77	0.84
Firmware	0.98	0.96	0.96	0.97	**1.00**	0.97
Guest OS	**0.93**	**0.93**	0.90	0.89	0.90	0.91
Host OS	0.88	0.85	0.88	**0.91**	0.80	0.86
Hypervisor	0.90	0.90	0.88	0.86	**0.95**	0.90
Phys. Hardware	0.85	0.90	0.88	0.91	**0.92**	0.89
Auth. Bypass	0.95	**0.96**	**0.96**	0.92	0.89	0.94
Code Execution	0.96	0.97	0.91	**0.98**	0.80	0.92
Context Escape	0.96	**0.99**	0.94	**0.99**	0.89	0.95
MitM	0.95	0.95	0.95	**0.98**	0.97	0.96
Trust Failure	0.96	0.95	**0.97**	0.94	0.90	0.94
Indirect Dis.	0.87	0.89	0.90	0.92	**0.93**	0.90
Privilege Esc.	0.84	0.86	**0.86**	0.85	0.77	0.84
Read	0.91	**0.93**	0.90	0.84	0.66	0.85
Resource Rem.	0.98	0.97	**0.99**	0.95	0.84	0.95
Service Int.	0.93	0.93	0.95	**1.00**	0.85	0.93
Write	**0.97**	0.95	0.95	0.91	0.56	0.87
ASLR	0.94	**0.97**	0.94	0.95	0.88	0.94
HPKP/HSTS	0.97	0.97	**0.99**	0.96	0.75	0.93
MFA	**0.97**	0.96	0.93	0.93	0.77	0.91
Phys. Security	0.97	**0.99**	0.97	0.97	0.95	0.97
Sandboxed	0.96	**0.99**	0.95	0.93	0.87	0.94
AVG F1 SCORE	0.91	0.92	0.91	0.91	0.85	

Table 5.2: F1-score values for each attribute using only Okutan's original dataset.

Attributes	**BERT**	**DistilBERT**	**XLNet**	**DeBERTa**	**Llama 2**	**AVG**
Limited Remote	0.82	0.78	0.71	0.87	**0.89**	0.81
Local	0.89	**0.91**	0.78	0.86	0.88	0.86
Remote	0.94	**0.96**	0.94	0.95	0.89	0.93
Physical	0.89	0.95	0.44	0.95	**1.00**	0.82
Application	**0.95**	0.93	0.87	0.83	0.85	0.88
Channel	**0.84**	0.80	0.76	0.62	0.80	0.77
Firmware	0.94	0.96	0.96	0.96	**1.00**	0.96
Guest OS	**0.91**	0.87	0.85	0.87	0.83	0.86
Host OS	**0.91**	0.87	0.93	0.86	0.67	0.85
Hypervisor	**1.00**	**1.00**	0.97	0.90	0.93	0.96
Phys. Hardware	0.87	**0.88**	0.81	**0.88**	0.67	0.82
Auth. Bypass	0.95	0.89	0.86	0.90	**1.00**	0.92
Code Execution	**1.00**	**1.00**	0.97	0.95	0.97	0.98
Context Escape	**0.97**	**0.97**	**0.97**	0.88	0.91	0.94
MitM	**1.00**	**1.00**	0.98	0.95	**1.00**	0.99
Trust Failure	0.97	0.90	0.93	0.93	**1.00**	0.95
Indirect Dis.	**1.00**	**1.00**	**1.00**	0.97	0.92	0.98
Privilege Esc.	0.97	0.97	0.97	**1.00**	0.94	0.97
Read	0.93	0.98	0.94	0.86	0.65	0.87
Resource Rem.	**1.00**	**1.00**	0.97	0.96	0.86	0.96
Service Int.	**1.00**	**1.00**	**1.00**	0.94	0.95	0.98
Write	**0.98**	0.93	0.89	0.96	0.63	0.88
ASLR	**0.98**	0.93	**0.98**	0.97	**0.98**	0.96
HPKP/HSTS	0.97	0.97	0.97	**0.98**	0.97	0.97
MFA	**0.96**	0.92	0.92	0.89	0.92	0.92
Phys. Security	**1.00**	**1.00**	0.95	0.89	0.95	0.96
Sandboxed	**0.96**	**0.96**	**0.96**	0.93	**0.96**	0.96
AVG F1 SCORE	0.95	0.94	0.90	0.90	0.89	

f1 score of models such as XLNet improves from 0.76 to 0.87 with the expanded dataset. However, it's worth noting that some models like DistilBERT see a slight decrease in f1 score, suggesting variability in how different models adapt to the increased data size.

The Firmware attribute shows a consistent performance by Llama 2, achieving a perfect score of 1.00 in both datasets. This consistency across datasets emphasizes the model's robustness in identifying firmware vulnerabilities, irrespective of the dataset size. For Guest OS vulnerabilities, the expanded dataset brings slight improvements or maintains performance for most models, with Llama 2's f1 score increasing from 0.83 to 0.90.

The Host OS attribute sees a mixed impact, with BERT's f1 score improving from 0.91 using only Ahmet et al.'s data to 0.88 with the expanded dataset. In the Hypervisor vulnerabilities classification, the original dataset sees a perfect score of 1.00 for both BERT and DistilBERT, which slightly decreases with the expanded dataset. Lastly, Physical Hardware vulnerabilities classification shows an improvement in accuracies for models like BERT and DistilBERT with the expanded dataset.

Impact Method

Authentication Bypass sees a shift in performance. The expanded dataset shows high accuracies with DistilBERT at 0.96 and Llama 2 slightly lower at 0.89. Interestingly, using only the original dataset, Llama 2 achieves a perfect score of 1.00. The Code Execution vulnerabilities illustrate a small difference with the original dataset alone, where BERT and DistilBERT both hit perfect accuracies of 1.00. Compared to the expanded dataset's highest f1 score of 0.98 by DeBERTa V3.

Context Escape classification reveals that the expanded dataset aids in achieving higher accuracies for models like DistilBERT at 0.99. However, the original dataset still supports strong performance, with all models achieving high f1 score, peaking at 0.97. Man-in-the-Middle vulnerabilities present a scenario where the original dataset enables perfect scores of 1.00 for BERT, DistilBERT, and Llama 2, surpassing the expanded dataset's highest f1 score of 0.98 by DeBERTa V3. Similarly, Trust Failure shows the original dataset drives Llama 2 to achieve a perfect f1 score of 1.00, whereas, in the expanded dataset, the highest f1 score is 0.97 by XLNet.

Logical Impact

The Indirect Disclosure sees a improvement in models trained on the original dataset, with BERT, DistilBERT, and XLNet each achieving perfect scores of 1.00, significantly higher than the combined dataset's top score of 0.93 by Llama 2. For Privilege Escalation, the original dataset again shows superior model performance, with DeBERTa V3 reaching a perfect f1 score of 1.00 and other models like BERT and DistilBERT not far behind at 0.97. Compared to the combined dataset, where accuracies peak at 0.86, the original dataset appears to offer a more concise representation of Privilege Escalation vulnerabilities, facilitating better model training and classification.

In the Read attribute, models trained on the original dataset show strong performance, especially DistilBERT at 0.98, surpassing the combined dataset's highest f1 score of 0.93 by BERT. Similarly, resource Removal vulnerabilities demonstrate a significant advantage for models trained on the original dataset, with BERT and DistilBERT achieving perfect scores of 1.00. In Service Interrupt, the original dataset approach perfect accuracies, with BERT and DistilBERT each scoring 1.00. Lastly, the Write attribute presents a scenario where the original dataset and the combined dataset perform very closely, such as BERT's 0.98 in the original dataset, compared to the combined dataset's best of 0.97 by BERT.

Mitigation

The ASLR (Address Space Layout Randomization) shows higher accuracies with BERT and XLNet (0.98), compared to those trained on the combined dataset, with DistilBERT reaching 0.97 and Llama 2 at 0.88. The HPKP/HSTS (HTTP Public Key Pinning/HTTP Strict Transport Security) presents a scenario where models trained on both datasets achieve high accuracies, but the combined dataset sees a slight dip for Llama 2 to 0.75.

In MultiFactor Authentication, the combined dataset maintains strong performance across most models, with BERT and DistilBERT hitting 0.97 and 0.96, respectively, compared to the original dataset's performance, where accuracies like Llama 2's reach 0.92. The Physical Security classifications demonstrate that the combined dataset enables models to achieve high accuracies, with DistilBERT and Physical Security both at 0.99. Likewise, models trained solely on the original dataset, such as BERT, attain perfect scores of 1.00. Lastly, for Sandboxed environments, the overall performance remains consistently high across both datasets. The accuracies across models like DistilBERT at 0.99 (combined dataset) versus 0.96 (original dataset).

Average F1 Scores of Noun Groups Attributes

The extension of datasets generally resulted in varied performance across these different attributes. For instance, in the Attack Theater attribute, more generalized attributes such as Limited Remote, Local, and Remote exhibited notable performance decreases with augmented dataset scores of 0.67, 0.77, and 0.83 respectively, compared to the original dataset scores of 0.81, 0.86, and 0.93. This suggests potential misalignment or the introduction of noise from the additional data within these specific attributes.

Conversely, improvements were observed in areas requiring high levels of specificity such as Firmware in the Context attribute and Physical Security in Mitigation, where F1 scores increased to 0.97. This enhancement suggests that the additional data included valuable and relevant examples, enhancing the models' abilities to accurately classify these complex vulnerabilities.

However, the Impact Method attribute displayed mixed results. While the attributes Code Execution and Context Escape showed slight decreases in performance in the augmented dataset

with scores of 0.92 and 0.95, respectively, they did not match the original dataset's scores of 0.98 and 0.94. This indicates that although the augmented data provided more diverse examples, it might not have consistently enhanced the model's ability to generalize across different vulnerability contexts.

The Logical Impact attribute also demonstrated varied impacts. Some areas like Resource Removal maintained robust performance with an F1 score of 0.95, but others such as Read and Write underperformed with F1 scores of 0.85 and 0.87 compared to the original dataset's scores of 0.87 and 0.88. These discrepancies underscore the challenges in dataset augmentation, where not all additions uniformly benefit all types of attributes.

RQ1: Does the expansion of datasets lead to enhanced performance in vulnerability classification by large language models?

The extension of datasets for fine-tuning LLMs in software vulnerability classification incorporates recent data from 2021 to 2023. This update aimed to capture a broader array of vulnerabilities and reflect current cybersecurity trends. However, the new dataset did not uniformly enhance model performance across the five noun groups studied, indicating variability in the effectiveness of the added data. We conclude that the annotations' quality, provided by students, could have influenced the outcomes, highlighting the need for rigorous quality control in data annotation. This study lays the groundwork for further exploration into dataset extension to optimize model performance across a broader spectrum of software vulnerabilities.

5.2 RQ2: How do large language models impact the effectiveness of software vulnerability classification?

The second question seeks to evaluate the performance of LLMs. It explores the comparative performance of large language models (LLMs) and conventional machine learning models in the context of software vulnerability classification across five distinct vulnerability noun groups. The comparative analysis draws on the results compiled in Table 5.1, which details the F1 scores achieved by LLMs in our research with the combined dataset, and Table 5.3, presenting the outcomes from the study by Okutan [29] using conventional machine learning models. By analyzing the differences in F1 scores and performance patterns, this study aims to identify areas where LLMs either excel or require further improvement in the classification and mitigation of various software vulnerabilities.

Table 5.3: Okutan's F1-score values using entropy-based methods and conventional machine learning models.

Attributes	**KLD**	**CE**	**SVM**	**NB**	**DT**	**RF**	**Vote**	**AVG**
Limited Remote	**0.91**	**0.91**	0.79	0.75	0.87	0.67	0.88	0.83
Local	**0.92**	0.89	0.77	0.80	0.8	0.75	0.82	0.82
Remote	0.94	0.94	0.92	0.89	**0.97**	0.9	0.94	0.93
Physical	0.9	0.91	0.96	0.85	**0.98**	0.9	**0.98**	0.93
Application	0.80	0.79	**0.84**	0.74	0.7	0.68	0.87	0.77
Channel	0.76	0.78	0.82	0.69	0.82	0.7	**0.9**	0.78
Firmware	0.85	0.85	0.89	0.73	**0.95**	0.82	0.94	0.86
Guest OS	0.76	0.76	0.93	0.7	**0.95**	0.88	0.94	0.85
Host OS	0.88	**0.88**	0.84	0.76	0.76	0.81	0.84	0.82
Hypervisor	0.76	0.77	**0.95**	0.57	**0.95**	0.76	**0.95**	0.82
Phys. Hardware								
Auth. Bypass	0.88	0.86	0.84	0.85	**0.97**	0.62	0.92	0.85
Code Execution	**0.94**	0.94	0.9	0.88	0.93	0.81	**0.94**	0.91
Context Escape	0.94	0.93	0.79	0.84	**0.98**	0.76	0.94	0.88
MitM	0.96	0.96	0.96	0.82	0.99	0.78	**1.00**	0.92
Trust Failure	0.91	0.9	0.67	0.84	**0.92**	0.39	0.93	0.79
Indirect Dis.	**0.94**	**0.94**	0.88	0.83	**0.94**	0.82	0.91	0.89
Privilege Esc.	0.78	0.78	**0.88**	0.71	**0.88**	0.82	0.87	0.82
Read	0.86	0.85	0.94	0.71	0.92	0.87	**0.97**	0.87
Resource Rem.	**0.88**	0.87	0.78	0.83	0.84	0.64	0.84	0.81
Service Int.	0.86	0.86	0.89	0.84	**0.97**	0.74	0.95	0.87
Write	0.83	0.82	0.95	0.76	0.94	0.88	**0.96**	0.88
ASLR	0.91	0.92	0.94	0.74	**0.96**	0.83	**0.96**	0.89
HPKP/HSTS	0.77	0.77	0.84	0.71	**0.97**	0.79	0.92	0.82
MFA	0.78	0.76	0.85	0.76	**0.92**	0.71	**0.92**	0.81
Phys. Security	0.89	0.88	0.92	0.78	**0.96**	0.85	**0.96**	0.89
Sandboxed	0.73	0.73	0.88	0.78	**0.93**	0.72	0.92	0.81
AVG F1 SCORE	0.86	0.86	0.87	0.78	0.91	0.78	0.92	

5.2.1 RQ1.1: How do models perform when classifying 27 vulnerability attributes?

This sub-question aims to compare different transform-based vs. conventional models in accurately classifying software vulnerabilities attributes, providing a comprehensive overview of their respective strengths and limitations in vulnerability classification.

For each of the 27 attributes:
Sum the F1 scores of the N models and divide by N to get the average F1 score.

Attack Theater

In the classification of software vulnerabilities, the performance of language model architectures varies significantly across different vulnerability contexts. In the scenario of Limited Remote vulnerabilities, there is a notable discrepancy in F1 scores among models. BERT achieves a lower F1 score of 0.60, whereas Llama 2 reaches 0.79. However, both scores are below the benchmark F1 score of 0.91 set by traditional models, such as KLD and CE. This comparison not only highlights the superior precision of traditional models in this attribute but also illustrates the variability within machine learning models, as evidenced by the Random Forest (RF) model's F1 score of 0.67.

Transitioning to Local vulnerabilities, the F1 scores of advanced transformer models like XLNet and DeBERTa V3 are competitive at 0.81, surpassing some traditional models. However, they do not exceed the peak F1 score of 0.92 achieved by KLD. Notably, Llama 2 exhibits a lower performance, with an F1 score of 0.69, indicating inconsistency in advancements across different language models.

For Remote vulnerabilities, DistilBERT registers an F1 score of 0.87, demonstrating substantial efficacy. Nonetheless, this is still outperformed by traditional models, where KLD and CE reach F1 scores of 0.94, and Decision Tree (DT) models achieve a notably higher score of 0.97. This underscores the ongoing relevance and superior capability of conventional models to discern nuances in Remote vulnerabilities with higher accuracy.

In the Physical vulnerability attribute, DistilBERT markedly excels with a perfect F1 score of 1.00, surpassing the F1 scores of traditional models like DT and Vote, both at 0.98. This exceptional performance indicates a potential unique strength of language models in precisely identifying Physical vulnerabilities, thereby challenging the dominance of conventional models in this area.

Context

In the Context noun group, Llama achieved remarkable f1 score in the Application and Firmware attributes, reaching 0.98 and 1.00 respectively, significantly exceed the best-performing conventional models, with Application vulnerabilities' highest conventional f1 score being 0.87 by Vote and Firmware's at 0.95 by both DT and Vote. Conversely, in the Channel attribute, where the highest

LLM f1 score by XLNet is slightly below Vote's 0.9, the data suggest areas where traditional models, particularly ensemble methods, maintain competitive advantage.

The analysis extends to the Guest OS and Host OS attributes, where LLMs closely match or slightly surpass the accuracies of conventional models. For Guest OS vulnerabilities, the LLMs, particularly BERT and DistilBERT, achieve up to 0.93 f1 score, which is competitive with the 0.95 achieved by SVM and DT among conventional models. In the Host OS attribute, DeBERTa V3's performance at 0.91 marginally exceeds the conventional high of 0.88 by KLD and CE. Futhermore, Hypervisor vulnerabilities further highlight the evolving capabilities of LLMs, with Llama 2 achieving an f1 score of 0.95, which aligns with the top performance among conventional models. This attribute exemplifies the narrowing gap between the newest generations of language models and established machine learning techniques, signaling a shift towards the increasing viability of LLMs in accurately identifying and categorizing complex software vulnerabilities.

Impact Method

In the Impact Method noun group, the performance comparison between language models (LLMs) and conventional models across five vulnerability attributes shows varied results. Starting with the Authentication Bypass attribute, LLMs like DistilBERT and BERT achieve high F1 scores of 0.96 and 0.95, respectively. While these scores are close, they do not surpass the highest score achieved by conventional models, which is 0.97 by the Decision Tree (DT) model. The lowest F1 score among conventional models is a 0.62 by the Random Forest (RF), underscoring a notable disparity in their performance.

For Code Execution vulnerabilities, LLMs demonstrate robust performance, particularly DeBERTa V3, which achieves an F1 score of 0.98. This score is higher than the best conventional model score of 0.94, shared by KLD, CE, and Vote. In the Context Escape attribute, LLMs such as DistilBERT and DeBERTa V3 reach remarkable F1 scores of 0.99, marginally exceeding the conventional model's highest score of 0.98 by DT.

In the attribute of Man-in-the-Middle vulnerabilities, the conventional Vote model achieves a perfect score of 1.00, which slightly exceeds the highest LLM F1 scores of 0.98 by DeBERTa V3 and Llama 2. Lastly, in the Trust Failure attribute, LLMs like BERT and XLNet achieve scores of 0.96 and 0.97, respectively, demonstrating their effectiveness. Among conventional models, DT scores 0.92 and Vote achieves 0.93, indicating strong competition, although RF markedly lags with an F1 score of 0.39.

Logical Impact

In the Logical Impact noun group, the evaluation is performed across various vulnerability classes such as Indirect Disclosure, Privilege Escalation, Read, Resource Removal, Service Interrupt, and Write. For Indirect Disclosure, LLMs, particularly Llama 2 with a 0.93 f1 score, approach the

benchmarks set by conventional models, where KLD and CE lead at 0.94. In the Privilege Escalation attribute, LLMs exhibit a varied performance, with their highest f1 score being 0.86 by both DistilBERT and XLNet. This is in contrast to the conventional models where SVM and DT reach a higher f1 score of 0.88. Notably, Llama 2's lower score of 0.77 points to the challenges LLMs face in consistently outperforming traditional models in privilege escalation scenarios.

The Read vulnerability sees DistilBERT achieving a 0.93 f1 score, lower than the conventional models' highest f1 score of 0.97 by Vote. Resource Removal vulnerabilities showcase a strong showing from LLMs, with XLNet's 0.99 f1 score surpassing all conventional model performances, with the closest being 0.88 by KLD and CE. This indicates a pronounced advantage of LLMs in classifying vulnerabilities that lead to the removal of system resources, though Llama 2's drop to 0.84 suggests not all LLMs are equally effective in this context.

Service Interrupt classification further highlights the potential of LLMs, with DeBERTa V3 achieving a perfect score of 1.00, outperforming the conventional high of 0.97 by DT. This demonstrates LLMs' superior understanding and classification capability for vulnerabilities that could cause service interruptions, despite Llama 2's lower f1 score of 0.85. In the Write attribute, LLMs like BERT demonstrate strong capabilities with a 0.97 f1 score, outperforming the conventional models' highest f1 score of 0.96 by Vote.

Mitigation

In the Mitigation noun group, the comparison between large language models (LLMs) and conventional machine learning (ML) models illustrates a trend where LLMs often outshine or closely compete with the traditional models in identifying effective mitigation strategies for various vulnerabilities. Starting with ASLR (Address Space Layout Randomization), LLMs such as DistilBERT achieve a remarkable 0.97 f1 score, surpassing the conventional models' best performance of 0.96 by DT and Vote.

For HPKP (HTTP Public Key Pinning)/HSTS (HTTP Strict Transport Security), LLMs display exceptional performance, with both BERT and DistilBERT hitting a 0.97 f1 score, and XLNet even higher at 0.99. This exceeds the highest f1 score of 0.97 by DT among conventional models. In MultiFactor Authentication, LLMs again demonstrate their prowess, with BERT and DistilBERT achieving 0.97 and 0.96 accuracies, respectively. These scores are notably higher than the best conventional model performance of 0.92 by both DT and Vote.

Physical Security classifications see LLMs like DistilBERT reaching a near-perfect f1 score of 0.99, closely matched by BERT and XLNet at 0.97. This performance is on par with or surpasses the conventional models, where DT and Vote again lead with 0.96. Lastly, in classifying Sandboxed environments, DistilBERT stands out with a 0.99 f1 score, significantly ahead of the conventional models' best score of 0.93 by DT.

This analysis underscores that while conventional models hold a slight edge in certain scenar-

ios within Attack Theater, LLMs demonstrate a robust capability across a broader spectrum of vulnerability attributes, particularly in Context, Impact Method, and Mitigation groups. This detailed evaluation provides essential insights for advancing the capabilities of LLMs in software vulnerability classification and mitigation.

Average F1 Scores of Noun Group Attributes

In the Attack Theater noun group, LLMs underperformed compared to conventional models with averages of 0.67 for Limited Remote, 0.77 for Local, and 0.83 for Remote vulnerabilities, against higher averages of 0.83, 0.82, and 0.93 by Okutan [29], respectively. However, LLMs excelled in several areas within the Context group, outperforming conventional models with a notable margin in Application (0.92 vs. 0.77), Firmware (0.97 vs. 0.86), Guest OS (0.91 vs. 0.85), Host OS (0.86 vs. 0.82), and Hypervisor (0.90 vs. 0.82).

The Impact Method group also saw LLMs outshine conventional models in all attributes: Authentication Bypass (0.94 vs. 0.85), Code Execution (0.92 vs. 0.91), Context Escape (0.95 vs. 0.88), Man-in-the-Middle (0.96 vs. 0.92), and Trust Failure (0.94 vs. 0.79). Similarly, in the Logical Impact group, while performance was competitive, LLMs demonstrated superior results in Resource Removal (0.95 vs. 0.81) and Service Interrupt (0.93 vs. 0.87), although they slightly trailed in Read and Write attributes.

Lastly, the Mitigation group saw LLMs consistently outperform conventional models across all attributes: ASLR (0.94 vs. 0.89), HPKP/HSTS (0.93 vs. 0.82), MultiFactor Authentication (0.91 vs. 0.81), Physical Security (0.97 vs. 0.89), and Sandboxed environments (0.94 vs. 0.81). This overall superior performance in Mitigation underscores the potential of LLMs in evolving beyond conventional models for identifying effective mitigation strategies.

5.2.2 RQ1.2: Which models have the best average f1 score across 27 vulnerability attributes?

This sub-question aims to identify which individual models demonstrate the greatest overall effectiveness in accurately classifying software vulnerabilities attributes.

For each of the N models:

Sum the F1 scores of the N attributes and divide by N to get the average F1 score.

In an analysis comparing the performance of large language models (LLMs) and conventional machine learning models reported by Okutan [29], notable trends and capabilities of individual models are revealed across various vulnerability attributes. The LLMs examined include BERT, DistilBERT, XLNet, DeBERTa V3, and Llama 2, each demonstrating strengths in specific contexts.

DistilBERT emerged as the standout performer among the LLMs, showcasing consistent high scores across multiple attributes with an impressive average of 0.97 in Mitigation and a strong

presence in the Impact Method group with a 0.96 average. BERT also showed strong performance, particularly in Mitigation and Logical Impact, where its comprehensive capabilities are evident. XLNet and DeBERTa V3 displayed solid and consistent results, often matching or surpassing their LLM counterparts in many vulnerability attributes.

On the conventional models side, as reported by Okutan [29], the models include Kullback–Leibler (KLD), Cross-Entropy (CE), Support Vector Machine (SVM), Naive Bayers (NB), Decision Tree (DT), Random Forest (RF), and Vote. Among these, the Decision Tree (DT) and Ensemble Learning (Vote) models notably exhibited very competitive performances. Specifically, DT demonstrated exceptional capability in the Impact Method group, achieving an average score of 0.96, and also shone in the Mitigation group. The Vote model consistently provided strong scores across the board, achieving the highest average in some attributes, such as a perfect 1.00 in Man-in-the-Middle within the Impact Method group.

RQ2: How do large language models impact the effectiveness of software vulnerability classification?

The large language models (LLMs), particularly BERT and DistilBERT, stood out as the top performers in software vulnerability classification. BERT and DistilBERT achieved average F1 scores of 0.95 and 0.94 respectively using only the Okutan dataset, while DistilBERT achieved the highest performance with an average F1 score of 0.92 using the combined dataset. XLNet and DeBERTa V3 also demonstrated strong performances, each closely trailing with an average F1 score of 0.91 using the combined dataset. This underscores the strong capabilities of these models in software vulnerability classification. In comparison with Okutan's study, where only two out of seven machine learning models and entropy-based methods achieved an average F1 score higher than 0.90, LLMs outperformed the results of the models in that study. However, it's important to note that conventional Decision Tree model and the Ensemble Learning technique also displayed highly competitive performances with average F1 score of 0.91 and 0.92 respectively.

5.3 RQ3: What are the prevalent n-grams associated with each vulnerability attribute?

The third question shifts focus to the linguistic aspects, exploring the specific n-grams that are most prevalent and potentially indicative of different types of software vulnerabilities.

Attack Theater

The analysis of n-grams associated with software vulnerability category across various Attack Theater classifications reveals distinct linguistic patterns that align with each category's unique access requirements and attack vectors. In the Remote category, prevalent unigrams like "remote" and "chrome" and bigrams such as "google chrome" and "chrome prior" highlight vulnerabilities related to internet-based threats, particularly in web browsers like Google Chrome, emphasizing the importance of version updates. The Local vulnerabilities shift focus towards internal system threats with unigrams "code" and "execution," and bigrams "code execution" and "arbitrary code," indicating risks from unauthorized code executions accessible via local interfaces such as consoles or SSH. Limited Remote vulnerabilities, characterized by unigrams like "device" and bigrams "denial service" and "authentication bypass," reflect concerns over network-constrained attacks that combine elements of both remote and localized access. Lastly, the Physical category requires physical interaction, shown by dominant unigrams "physical" and "access," and the bigram "physical access," pointing to vulnerabilities that necessitate the attacker's physical presence.

Table 5.4: Top 3 prevalent n-grams in Attack Theater noun group categories.

Category	**Unigrams**	**Count**	**Bigrams**	**Count**
Limited Rmt	access	81	denial service	25
	device	54	authentication bypass	19
	prior	53	netgear devices	19
Local	code	100	code execution	72
	execution	97	arbitrary code	57
	arbitrary	69	successful exploitation	35
Physical	access	165	physical access	100
	physical	142	mysql cluster	38
	device	82	access device	27
Remote	prior	156	google chrome	86
	remote	118	chrome prior	75
	chrome	96	html page	66

Context

The distribution of N-grams across different software contexts underscores specific vulnerability concerns aligned with the functionality of each system layer. In Application, the frequent unigrams "remote" and "exploit" coupled with bigrams like "denial service" and "execute arbitrary" highlight the vulnerability to external attacks and unauthorized actions, reflecting the widespread concern for security in software applications that operate across diverse environments. The Channel category, with unigrams such as "channel" and "information" and bigrams like "information disclosure" and "communication channel," brings to light issues in data transmission channels, emphasizing the

potential for data breaches in logical communication setups, particularly where encryption and protocol integrity are compromised.

Moving deeper into system layers, Firmware vulnerabilities are prominent, with a high occurrence of the unigram "firmware" and bigrams such as "firmware prior" and "firmware version," pointing to the risks associated with outdated or unsecured firmware integral to the functioning of critical devices. Contrastingly, Guest OS and Host OS showcase focused concerns, with "guest os" and "issue fixed" being significant phrases, indicating vulnerabilities specific to virtualized environments and primary operating systems respectively. This delineates the differing levels of security prioritization between isolated virtual operating systems and the broader host systems that support them. Hypervisor and Physical Hardware categories further this narrative; "hypervisor" and "hardware" as key unigrams and "denial service" and "successful attacks" as notable bigrams in each category respectively emphasize the criticality of managing virtual resource allocation securely and protecting physical components from direct attacks.

Table 5.5: Top 3 prevalent n-grams in Context noun group categories.

Category	**Unigrams**	**Count**	**Bigrams**	**Count**
Application	remote	98	denial service	49
	exploit	93	execute arbitrary	39
	oracle	77	remote attackers	32
Channel	channel	109	information disclosure	39
	information	69	communication channel	33
	issue	53	protection relays	32
Firmware	firmware	288	firmware prior	75
	prior	92	firmware version	63
	version	75	series software	36
Guest OS	guest	226	guest os	117
	os	128	denial service	100
	service	101	cause denial	72
Host OS	issue	129	issue fixed	57
	kernel	90	addressed improved	55
	memory	73	issue addressed	41
Hypervisor	hypervisor	130	denial service	61
	guest	113	cause denial	46
	service	65	guest os	32
Phys. Hardware	hardware	112	mysql cluster	51
	access	79	successful attacks	26
	cluster	63	denial service	26

Impact Method

For Authentication Bypass noun group, prevalent terms such as "authentication" and "bypass" are prominently discussed, underlining the security risks associated with attackers bypassing identity verification systems to gain unauthorized access. This is particularly critical as it highlights the frequent attempts to undermine security through deceptive means, making it a key area of concern for maintaining strong authentication protocols. Similarly, Code Execution is frequently mentioned with terms like "code" and "execution," where bigrams such as "arbitrary code" signify the serious threats posed by attackers executing unauthorized code to compromise system integrity.

On another front, Context Escape uses "sandbox" and "escape," with the common bigram "sandbox escape," pointing to vulnerabilities that enable attackers to move from restricted to less controlled environments, exploiting these transitions to escalate privileges or access sensitive areas. Man-in-the-Middle (MitM) and Trust Failure vulnerabilities are highlighted by terms like "attack," "server," "key," and "encryption," with relevant bigrams indicating the interception of communications and exploitation of trust mechanisms, such as "sensitive information" and "encryption key." These insights not only delineate the specific vulnerabilities but also emphasize the critical areas where security efforts should be concentrated to mitigate the risks associated with each type of impact method.

Table 5.6: Top 3 prevalent n-grams in Impact Method noun group categories.

Category	**Unigrams**	**Count**	**Bigrams**	**Count**
Auth. Bypass	authentication	299	authentication bypass	172
	bypass	217	bypass authentication	141
	access	77	netgear devices	61
Code Execution	code	267	code execution	172
	execution	190	arbitrary code	141
	arbitrary	53	execute arbitrary	61
Context Escape	sandbox	198	sandbox escape	90
	escape	129	google chrome	71
	prior	85	html page	69
MitM	attack	89	sensitive information	36
	server	87	man middle	28
	certificate	81	obtain sensitive	19
Trust Failure	key	245	encryption key	40
	encryption	134	sensitive information	19
	keys	68	issue discovered	18

Logical Impact

The N-gram distribution within the Logical Impact noun group uncovers patterns tied to the severity of different impacts on system security. Indirect Disclosure is heavily characterized by terms like

"information" and "disclosure," reflecting the vulnerability that allows attackers to glean system information indirectly, which might not involve direct data breaches but still poses substantial risks to confidentiality. For Privilege Escalation, the prevalent use of "privilege" and "escalation" indicates the threat of attackers gaining higher system permissions than authorized, leading potentially to widespread system compromise.

Read and Write impacts are underscored by terms like "read," "write," "access," and "files," with bigrams "read write" and "bounds write" suggesting unauthorized data access and modification risks that breach data confidentiality and integrity. Resource Removal focuses on the terms "files" and "arbitrary," indicating unauthorized data deletion threats. Lastly, Service Interruption is marked by "service" and "denial," highlighting disruptions to service availability caused by unauthorized actions, a critical aspect of maintaining operational reliability and trust in systems.

Table 5.7: Top 3 prevalent n-grams in Logical Impact noun group categories.

Category	**Unigrams**	**Count**	**Bigrams**	**Count**
Indirect Dis.	information	270	information disclosure	178
	discloure	197	execution privileges	45
	needed	89	privileges needed	45
Privilege Esc.	privilege	137	escalation privilege	99
	escalation	125	local escalation	51
	access	109	execution privileges	50
Read	read	198	read write	49
	access	120	information disclosure	38
	files	92	arbitrary files	31
Resource Rem.	files	182	arbitrary files	105
	arbitrary	157	remote attackers	41
	remote	98	attackers arbitrary	38
Service Int.	service	179	denial service	150
	denial	150	cause denial	61
	cause	102	remote cause	30
Write	write	198	read write	49
	access	104	bounds write	44
	file	102	possible bounds	38

Mitigation

In the mitigation strategies for software vulnerabilities, N-grams within each category highlight the focused security measures employed to protect systems. ASLR (Address Space Layout Randomization) is represented with unigrams like "buffer" and "overflow", pointing to the mitigation of memory corruption vulnerabilities through randomizing memory addresses, thus complicating exploitation attempts like arbitrary code execution. This strategy is essential for protecting against exploits that rely on predictable memory address knowledge. Similarly, HPKP/HSTS focuses on

enhancing web communication security, indicated by unigrams "http" and "request", ensuring that data transmitted over the web remains secure against interception and manipulation through strict transport security protocols or public key pinning.

Multi-Factor Authentication (MFA) is highlighted by "access" and "credentials," with bigrams like "access control," emphasizing the role of multiple authentication layers in preventing unauthorized access. Physical Security underscores the importance of tangible security measures, with "physical" and "device" being prominent, and "physical access" as a key bigram, which stresses controlling physical access to devices to prevent direct manipulation or data theft. Lastly, the Sandboxed environment, identified by "sandbox" and "java," with bigrams such as "java se," represents the isolation of processes to minimize the risks of system-wide compromises from potentially malicious code.

Table 5.8: Top 3 prevalent n-grams in Mitigation noun group categories.

Category	**Unigrams**	**Count**	**Bigrams**	**Count**
ASLR	buffer	192	buffer overflow	133
	overflow	161	arbitrary code	53
	code	90	denial service	40
HPKP/HSTS	http	257	http request	87
	oracle	140	accessible data	51
	request	138	successful attacks	43
MFA	access	173	mysql server	25
	credentials	153	access control	21
	server	89	successful exploit	21
Physical Security	access	192	physical access	88
	physical	136	mysql cluster	51
	device	110	physically proximate	38
Sandboxed	sandbox	158	java se	85
	java	137	google chrome	50
	code	110	access device	48

RQ3: What are the prevalent n-grams associated with each vulnerability attribute?

N-grams help reveal patterns and language correlations that might not be immediately apparent. They can uncover subtle cues that aid in understanding the nature of vulnerabilities. By analyzing n-grams, researchers and security practitioners can gain deeper insights into how different vulnerabilities manifest in language, which can inform the development of more targeted and effective mitigation strategies. Additionally, n-gram analysis can reveal evolving trends and new types of vulnerabilities, helping to stay ahead of emerging threats in cybersecurity.

CHAPTER 6
THREATS TO VALIDITY

Internal Validity Threats

1. *Student Annotators*: The primary internal validity threat arises from the reliance on student annotators, whose experience may not align with the complexity required for accurate vulnerability assessment.

2. *Bias and Inconsistencies*: Potential bias and inconsistencies in annotations could skew the data, impacting the reliability of the study.

To mitigate these issues, a rigorous training process was implemented based on established standards, and annotation sessions were limited to several hours per day to minimize fatigue. Furthermore, to enhance the depth and accuracy of annotations, we implemented peer-review sessions where disagreements could be discussed and resolved, ensuring a consensus was reached before finalizing the data. An additional proposed mitigation is periodic professional reviews to cross-verify annotations with experts, providing an external validation layer to improve data quality.

External Validity Threats

1. *Generalizability of a Small Dataset*:A significant threat to external validity is the use of a relatively small dataset, which may not adequately represent the broader spectrum of vulnerabilities encountered in more extensive systems. This limitation could affect the generalizability of the research findings.

2. *National Vulnerability Database*: The NVD's known issues, such as chronological inconsistencies, incomplete coverage, duplication of vulnerabilities, and poorly documented data fields—as highlighted by Ozment [31]—underscore the need for meticulous source verification and data curation in our study to avoid these pitfalls.

The initial mitigation involved carefully selecting a diverse set of vulnerabilities to annotate, aiming to cover different complexity levels and types. To further enhance the representativeness of our study, it would be beneficial to expand the dataset to include vulnerabilities from various sources beyond the NVD, thus broadening the scope and applicability of our findings. Consulting with professional security analysts and integrating their feedback into the dataset could also help bridge the gap between academic findings and practical applications.

Construct Validity Threats

1. *Interpretation of the Vulnerability Description Ontology*: The construct validity of this study could be compromised by the potential discrepancies in how vulnerability characteristics are interpreted under the VDO framework, particularly with the multi-label CVE approach.

2. *Confidence Scoring System*: The use of a scoring system to measure annotators' confidence might not capture the exact understanding required for certain vulnerabilities.

Regular updates to the annotation procedures have been made to address these issues, and a cross-validation approach with cybersecurity experts is proposed to ensure the annotations accurately reflect the CVE characteristics. Additionally, ongoing reviews and adaptations of the annotation methodologies based on feedback and observed challenges will help maintain the precision of the dataset. To further strengthen construct validity, conducting experiments with larger and more diversified CVE sets in future studies could help verify and refine our findings.

CHAPTER 7
CONCLUSION

Our study provides a comprehensive analysis of the impact of Large Language Models (LLMs) on software vulnerability classification, along with the effectiveness of dataset expansion.

1. Firstly, our findings suggest that while the expansion of the dataset aimed to capture a broader array of vulnerabilities, the added data did not uniformly enhance model performance, indicating the critical role of annotation quality.

2. Secondly, LLMs, particularly BERT and DistilBERT, stood out as the top performers in software vulnerability classification. XLNet and DeBERTa V3 also demonstrated strong performances, underscoring the capabilities of these models in software vulnerability classification. In comparison with Okutan's study, LLMs outperformed the results of the models in that study. However, we must note that the conventional Decision Tree model and the Ensemble Learning technique also displayed highly competitive performances.

3. Lastly, n-grams analysis revealed patterns associated with vulnerability attributes, offering insights for targeted mitigation strategies and staying ahead of emerging threats.

These findings highlight the importance of not only applying advanced machine learning techniques but also maintaining rigorous quality control in data annotation to optimize the performance of models used in software vulnerability classification. This study lays the groundwork for further research into dataset extension and the integration of LLMs to refine the processes of vulnerability assessment in the field of cybersecurity.

BIBLIOGRAPHY

[1] Thamali Madhushani Adhikari and Yan Wu. Classifying software vulnerabilities by using the bugs framework. *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6, 2020.

[2] Catarina Araujo. Source code and data for software vulnerability classification. Online, April 2024. Available online at https://github.com/acatarinaoaraujo/software-vulnerability-classification/tree/main. Accessed: 2024-04-14.

[3] Harold Booth, Doug Rike, and Gregory A. Witte. The national vulnerability database (nvd): Overview. technical report. *National Institute of Standards and Technology (NIST)*, 2013.

[4] Harold Booth and Christopher Turner. Draft nistir 8138, vulnerability description ontology (vdo). technical report. *National Institute of Standards and Technology (NIST)*, 2016.

[5] Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Comput. Linguist.*, page 249–254, June 1996.

[6] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, pages 3280–3296, 2020.

[7] Qiuyuan Chen, Lingfeng Bao, Li Li, Xin Xia, and Liang Cai. Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures. *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 345–354, 2018.

[8] Yang Chen, Andrew E. Santosa, Ang Ming Yi, Abhishek Sharma, Asankhaya Sharma, and David Lo. A machine learning approach for vulnerability curation. *2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR)*, pages 32–42, 2020.

[9] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. *ICLR*, 2020.

[10] Kelley Dempsey, Paul Eavy, George Moore, and Eduardo Takamura. Automation support for security control assessments: Software vulnerability management. *National Institute of Standards and Technology (NIST)*, 2020.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *North American Chapter of the Association for Computational Linguistics*, 2019.

[12] Michael Fu and Chakkrit Tantithamthavorn. Linevul: A transformer-based line-level vulnerability prediction. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 608–620, 2022.

[13] Zeyu Gao, Hao Wang, Yuchen Zhou, Wenyu Zhu, and Chao Zhang. How far have we gone in vulnerability detection using large language models. 2023.

[14] Danielle Gonzalez, Holly Hastings, and Mehdi Mirakhorli. Automated characterization of software vulnerabilities. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 135–139, 2019.

[15] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. Learning to predict severity of software vulnerability using only vulnerability description. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 125–136, 2017.

[16] Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. 2021.

[17] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *International Conference on Learning Representations*, 2021.

[18] Guoyan Huang, Yazhou Li, Qian Wang, Jiadong Ren, Yongqiang Cheng, and Xiaolin Zhao. Automatic classification method for software vulnerability based on deep neural network. *IEEE Access*, 7:28291–28298, 2019.

[19] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. The importance of accounting for real-world labelling when predicting software vulnerabilities. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 695–705, 2019.

[20] Klaus Krippendorff. Reliability in content analysis: Some common misconceptions and recommendations. *Human Communication Research*, 30:411–433, 2004.

[21] David Last. Using historical software vulnerability data to forecast future vulnerabilities. *2015 Resilience Week (RWS)*, pages 1–7, 2015.

[22] Z. Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, pages 2244–2258, 2018.

[23] Z. Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. Vuldeelocator: A deep learning-based fine-grained vulnerability detector. *IEEE Transactions on Dependable and Secure Computing*, pages 2821–2837, 2020.

[24] Z. Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. 2018.

[25] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019.

[26] Fabio Massacci and Hung Nguyen. Which is the right source for vulnerability studies? an empirical analysis on mozilla firefox. *6th International Workshop on Security Measurements and Metrics, MetriSec 2010*, August 2010.

[27] Alejandro Mazuera-Rozo, Anamaria Mojica-Hanke, Mario Linares-Vásquez, and Gabriele Bavota. Shallow or deep? an empirical study on detecting vulnerabilities using deep learning. *IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 276–287, 2021.

[28] The Hacker News. Researchers discover lg smart tv vulnerabilities allowing root access. Online, April 2024. Available online at https://thehackernews.com/2024/04/researchers-discover-lg-smart-tv.html. Accessed: 2024-04-14.

[29] Ahmet Okutan, Peter Mell, Mehdi Mirakhorli, Igor Khokhlov, Joanna C. S. Santos, Danielle Gonzalez, and Steven Simmons. Empirical validation of automated vulnerability curation and characterization. *IEEE Transactions on Software Engineering*, 49(5):3241–3260, 2023.

[30] Marwan Omar and Darrell Burrell. From text to threats: A language model approach to software vulnerability detection. *International Journal of Mathematics and Computer in Engineering*, 2023.

[31] Andy Ozment. Vulnerability discovery software security. *University of Cambridge: Cambridge University Press*, 2007.

[32] Rebecca Passonneau. Computing reliability for co-reference annotation. *University of Cambridge: Cambridge University Press*, 2004.

[33] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[34] Ernesto Russo, Andrea Di Sorbo, Corrado Aaron Visaggio, and Gerardo Canfora. Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. *Journal of Systems and Software*, 156, June 2019.

[35] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. 2019.

[36] Benjamin Steenhoek, Md Mahbubur Rahman, Richard Jiles, and Wei Le. An empirical study of deep learning models for vulnerability detection. *Proceedings of the 45th International Conference on Software Engineering*, page 2237–2248, 2023.

[37] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. Transformer-based language models for software vulnerability detection. *Proceedings of the 38th Annual Computer Security Applications Conference*, page 481–496, 2022.

[38] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. 2023.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[40] Dumidu Wijayasekara, Milos Manic, and Miles McQueen. Vulnerability identification and classification via text mining bug databases. *Proceedings, IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pages 3612–3618, October 2014.

[41] Mark Williams, Roberto Camacho Barranco, Sheikh Motahar Naim, Sumi Dey, M. Hossain, and Monika Akbar. A vulnerability analysis and prediction framework. *Computers Security*, 92:101751, February 2020.

[42] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Neural Information Processing Systems*, 2019.

[43] Su Zhang, Xinming Ou, and Doina Caragea. Predicting cyber risks through national vulnerability database. *Information Security Journal: A Global Perspective*, 24(9):1–13, November 2015.

[44] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[45] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. *ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation*, pages 421–428, January 2010.