KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Transformer-Based Models for Code Vulnerability Detection

Automating Code Vulnerability Detection in GitHub Issues

DANIELE CIPOLLONE

Stockholm, Sweden, 2024

Transformer-Based Models for Code Vulnerability Detection

Automating Code Vulnerability Detection in GitHub Issues

DANIELE CIPOLLONE

Degree Programme in Computer Science and Engineering
Date: August 29, 2024

Supervisors: Changjie Wang, Akhila Rao
Examiner: Marco Chiesa
School of Electrical Engineering and Computer Science
Host company: Research Institute of Sweden, RISE
Swedish title: Transformatorbaserade modeller för Detektering av kodsårbarhet
Swedish subtitle: Automatisera kodsårbarhetsdetektering i GitHub issues

© 2024 Daniele Cipollone

Abstract

The rapid growth of open-source software has increased the importance of timely and accurate vulnerability detection. This thesis explores the use of transformer-based models and machine learning techniques to automate the identification of software vulnerabilities through the analysis of GitHub issues. In this work, a novel approach is proposed by defining a new dataset specifically for classifying GitHub issues that are relevant to the identification of vulnerabilities. Various classification methodologies, incorporating both Large Language Models and embedding models, are analyzed and compared. The final solution leverages both models to optimize computational costs while maximizing the quality of the results produced by the system. The effectiveness of this approach demonstrates its potential for real-world application in early vulnerability detection, which could significantly reduce the window of exploitation for software vulnerabilities. This research contributes to the field by providing a framework for automated vulnerability detection that is both computationally efficient and scalable, with implications for improving the security of open-source software ecosystems.

Keywords

Vulnerability Detection, GitHub Issues, Transformer Based Models, Large Language Models (LLM), Embedding Models

Sammanfattning

Den snabba tillväxten av programvara med öppen källkod har ökat vikten av snabb och korrekt upptäckt av sårbarheter. Det här examensarbetet utforskar användningen av transformatorbaserade modeller och maskininlärningstekniker för att automatisera identifieringen av mjukvarusårbarheter genom analys av GitHub-problem. Vi jämför flera tillvägagångssätt, inklusive zero-shot och few-shot-inlärning med stora språkmodeller (LLM) och inbäddningsbaserade modeller i kombination med XGBoost-klassificerare. Våra resultat tyder på att även om individuella modeller, såsom de som är baserade på inbäddningar eller LLM, ger värdefulla insikter, erbjuder en hybrid metod som kombinerar dessa metoder överlägsen prestanda. Hybridmodellen utnyttjar de effektiva och kostnadseffektiva klassificeringsmöjligheterna hos inbäddade modeller samtidigt som resultatens tolkningsbarhet och noggrannhet förbättras genom LLM. Studien tar också upp utmaningarna med att bearbeta stora datamängder. Effektiviteten av detta tillvägagångssätt visar dess potential för verklig tillämpning vid tidig upptäckt av sårbarheter, vilket avsevärt skulle kunna minska exploateringsfönstret för sårbarheter i programvara. Denna forskning bidrar till fältet genom att tillhandahålla ett ramverk för automatisk sårbarhetsdetektion som är både beräkningseffektivt och skalbart, med implikationer för att förbättra säkerheten för ekosystem med öppen källkod.

Nyckelord

Sårbarhetsdetektering, GitHub Issues, transformatorbaserade modeller, stora språkmodeller (LLM), inbäddningsmodeller

Acknowledgments

I would like to express my heartfelt gratitude to Marco Chiesa for granting me the wonderful opportunity to work on this incredible experience from which I have learned so much. I also wish to extend my thanks to Mariano Scazzariello, Changjie Wang, who have been consistently supportive throughout this journey and from whom I have gained invaluable insights.

Furthermore, I am thankful to RISE for hosting me and providing me with the chance to undertake this thesis project, and to my industrial supervisor Akhila Rao for this remarkable opportunity.

Stockholm, August 2024
Daniele Cipollone

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

AI	Artificial Intelligence
CNA	CVE Numbering Authorities
CoT	Chain-of-Thought
CTI	Cyber Threat Intelligence
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DL	Deep Learning
DNN	Deep Neural Network
LLM	Large Language Model
LLMs	Large Language Models
LM	Language Modeling
ML	Machine Learning
NIST	National Institute of Standards and Technology
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NVD	National Vulnerability Database
SVM	Support Vector Machine

Chapter 1

Introduction

Innovation and technological progress are leading us towards a comprehensive digitalization of our world. This digital transformation offers countless advantages, such as allowing the exchange of information at an unparalleled speed. However, this advancement is not without drawbacks, particularly related to the exposure to cybersecurity risks. It is easy to recognize that the development and consistent updating of software often leads to the potential presence of bugs and vulnerabilities. As such, there is a continuous process of software refinement and repair following the discovery of potential vulnerabilities.

Different methods are employed to detect vulnerabilities, which range from lower-level tests on the executed code to simulations of possible scenarios aimed at penetrating or disrupting the system. Despite these measures, it is still insufficient to prevent the presence of bugs or vulnerabilities in the final software. Among these, Zero-day vulnerabilities pose a particularly serious threat. These are undiscovered flaws in applications or operating systems, for which no defense or patch exists because the software manufacturer is unaware of their presence. As highlighted in Google's Threat Analysis Group's March 2024 report, there has been a marked increase in the exploitation of such vulnerabilities, underscoring the growing importance of proactive threat detection[1].

Traditionally, vulnerability detection has relied on a combination of expert analysis and Artificial Intelligence (AI) techniques to sift through vast amounts of text data, such as social media posts, darknet communications, and other indicators. These methods, while effective, are not without limitations. There is a significant opportunity for advancement in this field through the application of Large Language Model (LLM). These models have demonstrated exceptional abilities in text interpretation and identification of logical relationships within complex data sets, making them promising tools to improve vulnerability detection processes[2, 3].

This work explores the potential of LLM and embedding-based models to identify vulnerabilities through the analysis of communication channels, specifically focusing on GitHub Issues, as they constitute a significant portion of these communications. GitHub is a web-based platform for version control and collaborative software development. Leveraging the advanced text processing capabilities of these models, this work aims to determine whether it is feasible to identify software vulnerabilities in a cost-effective and scalable manner. The findings of this study not only demonstrate the viability of these approaches, but also highlight their potential to significantly enhance existing cybersecurity measures by enabling earlier detection of vulnerabilities, thus reducing the risk of exploitation as mentioned above.

1.1 Background

LLM are a type of AI model known for their strong performance in many Natural Language Processing (NLP) tasks[4]. AI aims to mimic human intelligence by interpreting data and performing tasks that usually require human thinking. These models are part of the broader field of Generative AI, which involves creating new content such as text, images, videos, and music [5]. Specifically, LLM focus on Language Modeling (LM) to predict the next word in a sequence based on the likelihood of previous words. This technology has been explored in various fields, including cybersecurity, where it has shown promise in the task of vulnerability detection that involves an understanding of the code's context [6, 7, 8].

The lifecycle of vulnerabilities can be divided into three phases: Black Risk, Gray Risk, and White Risk [9]. These stages cover the time from when a vulnerability is discovered, through its publication, to the countermeasures taken. The most critical phase, Black Risk, is the period between the discovery and disclosure of a vulnerability. Accurately pinpointing the moment a new vulnerability is discovered is challenging. According to [10], this moment is defined as "[...] the first date when a vulnerability is described on an information channel where the disclosed information on the vulnerability is (a) freely available to the public, (b) published by a trusted and independent channel, and (c) has undergone analysis by experts such that risk rating information is included." After a vulnerability is disclosed, it is published with details on the National Vulnerability Database (NVD). The later stages, Gray and White Risk, involve the release of security patches, reducing the threat as the vulnerability becomes known to both the software producer and its users.

Cyber Threat Intelligence (CTI) involves knowledge and information about potential vulnerabilities and attack methods in computer science. This field has grown significantly with the increase in cyber attacks. According to a 2024 IBM report, exploiting known vulnerabilities is a major attack vector [11]. Keeping up-to-date with vulnerabilities and sharing this information is crucial. Sharing typically happens through blogs, mailing lists, and social media. Analyzing these sources can provide a broader view of the vulnerabilities and sometimes offer early insights before official disclosures are published [12].

1.2 Problem

Given the ongoing digitization process and the increasing relevance of cybersecurity, it has become challenging to keep up with all sources of information. These communications can often come through social media profiles of individual developers, proprietary blogs of software houses, countless mailing lists, and so on. Consequently, staying updated on cybersecurity-related matters has become not just a difficult undertaking, but also a time-consuming task for both individuals and IT teams managing entire systems.

In this landscape, various companies provide a service for gathering information and producing regular security reports. These companies are primarily comprised of expert teams tasked with analyzing the sources and, through their knowledge, identifying potential security flaws. However, with a human component, this solution inevitably involves a longer operational time and possible room for human error in judgment.

An important question then arises: is it possible to automate this information analysis process using LLMs?

1.3 Purpose

This research aims to automate and speed up the process of analyzing cybersecurity-related information from sources like GitHub. By automating this analysis, we can help reduce the ”black risk phase,” which is the critical period between discovering a vulnerability and publicly disclosing it. This phase is especially vulnerable to exploitation.

In this context, LLM can play a crucial role, as they are capable of understanding and processing text from various sources. Our focus is specifically on GitHub issues, given its vast amount of data, to explore how LLM can be used to identify and assess vulnerabilities more efficiently and accurately.

1.4 Goals

The primary aim of this project is to explore and establish a robust method to automate vulnerability detection using advanced machine learning models, specifically focusing on analyzing GitHub issues. The project addresses several key objectives:

- Develop and Evaluate Classification Models: Investigate the effectiveness of various models, including transformer-based architectures for classifying GitHub issues related to vulnerabilities.
- Assess System Performance: Measure the performance of the implemented models using standard classification metrics. The goal was to understand how well these models could detect vulnerabilities compared to a baseline approach and evaluate their accuracy in providing detailed vulnerability descriptions.

These goals were addressed through a series of iterative experiments and evaluations, ultimately demonstrating the potential of machine learning models to improve vulnerability detection processes and provide valuable information for future research and application.

1.5 Research Methodology

This study employs an experimental research approach, characterized by exploration, experimentation, and evaluation phases. The methodology is designed to rigorously investigate the potential of LLM technology in the context of vulnerability detection using GitHub issues.

The research process begins with an extensive review of relevant literature, which provides a foundational understanding of current methodologies and identifies gaps in existing research. This review informed the design and development of our approach, ensuring that our methodology is grounded in established knowledge while addressing novel aspects of vulnerability detection.

Following the literature review, the focus shifts to data collection. Given the nature of the research, it was crucial to curate a dataset that accurately represents the relationship between GitHub issues and vulnerabilities. To this end, we utilized information from the NVD, a comprehensive database that correlates known vulnerabilities with associated issues. This dataset serves as the basis for supervised learning, where each issue is labeled to indicate its relevance to vulnerability identification.

The choice of methodologies, including the classification approaches applied to GitHub issues, was driven by the need to balance effectiveness and computational efficiency. Transformer-based models were selected for their advanced capabilities in text interpretation and contextual analysis. The experimental framework also includes a novel aspect of LLM usage, focusing on their ability to extract and interpret contextual information from GitHub issues.

In summary, the research methodology integrates a comprehensive literature review, meticulous data collection, and advanced classification techniques to address the challenges of vulnerability detection. The detailed execution of these methodologies is outlined in Chapter 3.

1.6 Ethics and Sustainability

This thesis also addresses ethical and sustainability aspects related to the research. The ability to identify vulnerabilities before traditional disclosure dates provides a significant advantage in preventing potential cyberattacks. Early detection of such vulnerabilities can enhance overall cybersecurity and mitigate risks more effectively.

Moreover, this thesis proposes methodologies that minimize the use of computationally and energetically intensive models. By employing simpler and more cost-effective models, we aim to reduce the environmental impact associated with high computational demands. This approach not only contributes to more sustainable research practices but also aligns with the broader goal of promoting ethical and eco-friendly technological advancements.

1.7 Delimitations

In this research, several boundaries and limitations were defined to focus the study and manage its scope effectively. The project exclusively utilized GitHub as the primary source of data for vulnerability detection. Although other information channels such as Twitter and mailing lists could offer valuable insights, these were beyond the scope of this research. This decision was driven by the extensive use and availability of data on GitHub, which provided a rich and relevant dataset for the purpose of evaluating the proposed methodologies. Additionally, the focus was specifically on models from OpenAI, excluding other potential alternatives like different transformer architectures or custom-built models. This selective approach was intended to streamline the experimentation and analysis, but limited the breadth of model evaluation.

Chapter 2

Background

This chapter aims to provide background information on the AI technologies used, introducing key concepts on NLP and its evolution over time, encompassing essential principles about Large Language Models (LLMs) and their operations. Further, it elucidates the process of identifying and processing software vulnerabilities.

2.1 Natural Language Processing

NLP is a subcategory of AI, which has gained increased recognition in recent years. Its primary objective revolves around endowing machines with the capacity to process information encoded in written text. This field represents a junction between computer scientists and other professionals, including linguists and philosophers. Language can be defined as a set of rules or an array of symbols combined and used for information transmission or broadcasting [13], thus portraying it as a powerful tool capable of embodying abstract concepts. NLP can be further subdivided into two main categories: Natural Language Understanding (NLU) and Natural Language Generation (NLG) [13]. NLU primarily focuses on extracting information from the text, such as concepts, emotions, entities, and so forth. Conversely, NLG concerns the generation of logically coherent text, capable of encapsulating information in natural language.

2.1.1 Recent developments

An important milestone in this field was achieved in the early 2000s, with the advent of the Neural Language Modeling mechanism. This method proposed by Bendigo et al. [14] involves the use of a feed-forward neural network to contemporaneously learn a representation for each word (similarity between words) and the probability function for sequences of words. In simpler terms, it proposed the concept of using a Neural Network to predict the probability of each subsequent word in the sequence provided to the model. Subsequently, in 2008, Collobert et al. [15] proposed a solution using Deep Neural Network (DNN) in a convolutional model. The paper suggested training the model on multiple types of NLP tasks concurrently. These tasks included part-of-speech tagging, chunking, named entity recognition, and semantic role labeling. The model was composed of the following layers: Lookup table layer, convolutional layer, max over-time pooling layer, and fully connected layers. In 2013, Mikolov et. al. [15] published a revolutionary methodology that significantly improved the generation of embeddings for word identification. This approach paved the way for realizing a more pragmatic relationship between words with similar semantic meanings or those logically correlated. An example provided in the paper demonstrates how their model established a relation between the names of states and their capitals. In 2014, Google presented a novel approach. They advocated for a method that shifted focus from individual words to sequences [16].

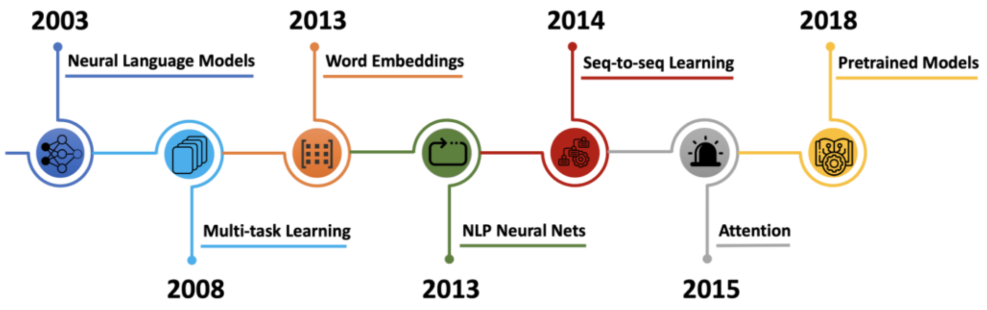
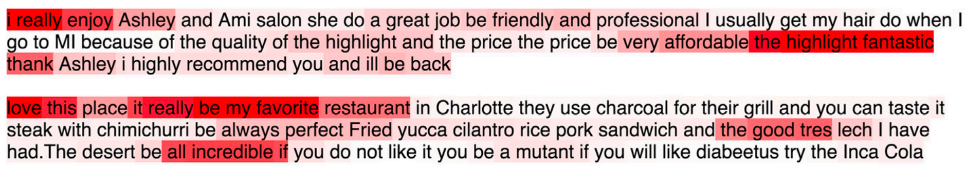


Figure 2.1: Evolution of NLP[17]

In 2015, a groundbreaking approach was proposed when Bahdanau et al. [18] introduced the concept of an ’attention’ mechanism, where networks can determine what to focus on relative to their current hidden state. Given the importance of this development in the progression of NLP, it is worth exploring the topic further.

2.1.2 Attention mechanisms

The attention mechanism has proven to be a pivotal moment in the field of Deep Learning (DL) and NLP. It allows us to optimize the inputs we provide to models, transmitting only truly effective information. This mechanism was first proposed as a solution to the translation of fixed- and limited-length text [18]. This approach diverges significantly from the fundamental encoder-decoder because it does not strive to encode an entire input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors, selectively choosing a subset of these vectors to decode the translation.



i really enjoy Ashley and Ami salon she do a great job be friendly and professional I usually get my hair do when I go to MI because of the quality of the highlight and the price the price be very affordable the highlight fantastic thank Ashley i highly recommend you and ill be back

love this place it really be my favorite restaurant in Charlotte they use charcoal for their grill and you can taste it steak with chimichurri be always perfect Fried yucca cilantro rice pork sandwich and the good tres lech I have had.The desert be all incredible if you do not like it you be a mutant if you will like diabeetus try the Inca Cola

Figure 2.2: Heatmap of attention from [19]

Consider the case in which our input is a list of words, attempts have been made over the years to represent word meanings in latent space with increasing precision [15]. However, the same words can have different meanings if placed in a different context. The aim of attention is therefore to establish a relationship between each specific word and the rest of the context, consequently modifying its representation in the latent space [20]. From its initial implementation, the attention mechanism has swiftly adapted for various tasks. The most popular attention mechanism implementation for LLM involves the use of a set of vectors, specifically Query, Key, and Value for each embedding. In practice, the attention function is computed on a set of queries simultaneously, all condensed into a matrix, Q. The keys and values are likewise collectively arranged into matrices, K and V respectively [21].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{2.1}$$

Two types of attention mechanisms can be identified: dot-product attention and additive attention. They differ on how the initial embedding is updated. Empirical findings have shown that for large values of d (which corresponds to the dimension of keys, values, and query), the additive method tends to outperform the multiplicative one [22].

2.2 Large Language Models

LLMs are defined as neural network models that have been trained on a large volume of text. Specifically, these models integrate two subgroups of AI - DL and NLP, with a focus on tasks related to the natural language generation. The foundation of LLMs is the Transformer architecture which is known for its scalability and adaptability to various tasks. Several private companies, notably OpenAI, are significantly contributing to the research on these models. One such contribution by OpenAI is making these models publicly available through ChatGPT. One of the primary characteristics of these models is the size as the number of parameters in these models often exceeds billions. Examples of such models are Generative Pre-trained Transformer (GPT) series and LLaMA 2 [23]. This extensive parameter count allows the models to capture a wide variety of patterns and links from the data they learn from, contributing towards improved performance in their associated tasks.

2.2.1 Deep Neural Networks

To gain a comprehensive understanding of what LLMs are, it is necessary to explore the neural networks that comprise this model. Deep Neural Networks are part of the subfield of DL, which in turn falls under the larger umbrella of Machine Learning (ML). At the core of this field is the concept of using neural networks not only to perform a given task based on features, but also to assign the machine the role of identifying these features, which may even be incomprehensible to humans. Contrary to conventional machine learning, where models are often requiring meticulous and specific feature engineering, deep learning represents a significant evolution due to its use of advanced representation learning methods. Deep learning models leverage architectures with multiple representation layers, known as deep neural networks, to automatically and hierarchically extract pertinent features from the data. This approach allows models to learn directly from raw data, eliminating the need for extensive domain knowledge [24]. These networks are inspired by the structure and connections of neurons within our brains. They are composed of an input layer where data are fed into the model, a series of hidden layers designed to extrapolate patterns, and an output layer where the format varies depending on the task to be performed [25]. Neurons are the basic component of these networks and are responsible for processing information. More specifically, they receive input signals, process them using a specific activation function, and propagate the signal through an output. They are interconnected with each other via weighted connections. The weights of these connections are known as parameters. The values associated with these parameters are calculated using the backpropagation algorithm during the model training phase.

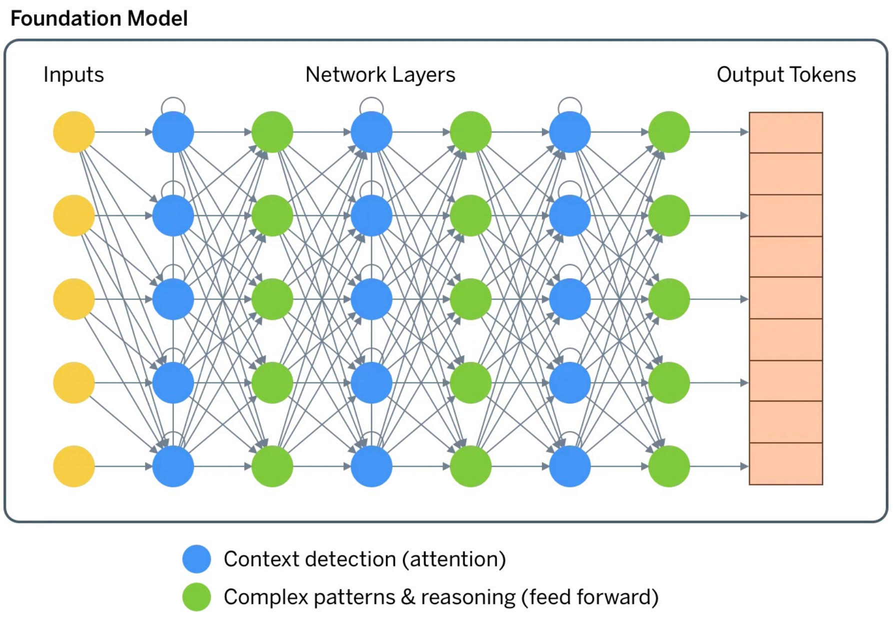

Foundation Model
Inputs
Network Layers
Output Tokens
Context detection (attention)
Complex patterns & reasoning (feed forward)

Figure 2.3: An illustration of the Transformer's architecture, reveals a multi-layered network with interconnected layers. [26]

2.2.2 Transformers

Building upon the concept of attention, a new model architecture known as Transformer was proposed in 2017[21]. Initially designed for tasks of translation, it has proven to be remarkably versatile, and adaptable to many tasks beyond the sphere of Natural Language Processing (NLP). This model was designed to optimize prior models that relied on the use of recurrent neural networks, which made them notably complex and difficult to train. Unlike their predecessors, Transformers are easily scalable and simpler to train due to their inherent parallelization capabilities. The original implementation follows an encoder-decoder structure. The encoder maps an input sequence to an internal representation, whereas the decoder yields a sequence of results, each produced in succession.

For a high-level analysis of this architecture, it is worth examining the

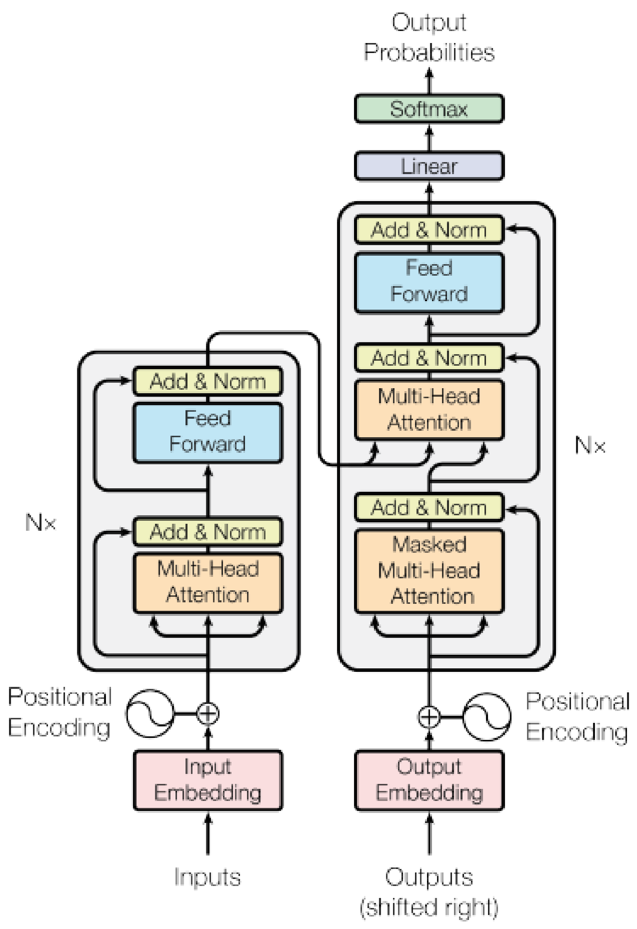


Figure 2.4: Transformer - model architecture from the paper “Attention is All You Need” [21]

various components that constitute its structure. Firstly, let’s consider the input and output embedding block, which maps each token to a unique vector, referred to as an ’embedding’. This vector embodies the meaning of the individual token. Each embedding is then coupled with a positional embedding, enabling the model to track the position of each token. Following this, there is a series of Multi-Head Attention and Normalization layers. The term ”Multi-Head” can be attributed to the mechanism whereby multiple attention layers operate simultaneously on independent linear projections of the input embeddings. Each of these attention heads, by focusing on different parts of the input, contributes unique contextual insights, a collective aggregation of which is remarkably effective at understanding complex patterns and relationships within the data. Moreover, the parallelizable nature of these layers significantly enhances the computational efficiency of the model. This model generates a probability distribution over the potential output vocabulary[21].

2.2.3 Encoder-only models

Encoder-only models are based upon the mechanism of transformers, enabling the representation of context and meaning in a dense vector form for a sequence of characters. This type of model has been found to be highly beneficial for a vast number of tasks within Natural Language Processing (NLP) as denoted by NLP. Among this genre of models, we encounter BERT [27], RoBERTa, and the text similarity engine from OpenAI [28]. The state-of-the-art for these types of models has been achieved through the extensive use of unsupervised learning on large datasets. Indeed, it has been demonstrated that models trained in this manner tend to outperform even those specialized in a single domain [28].

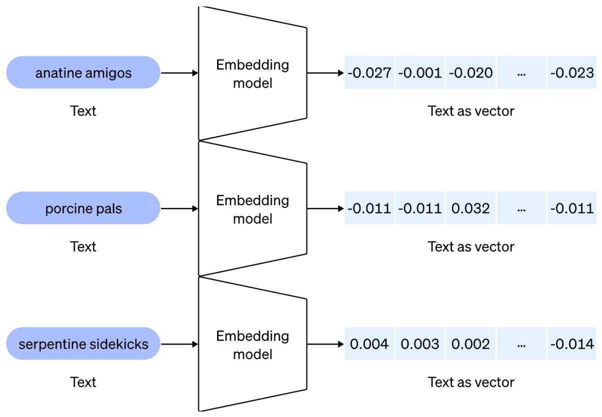


Figure 2.5: Embedding creation process. Image from OpenAI [29]

2.3 Security vulnerabilities

Security vulnerabilities are critical flaws or weaknesses in software systems that can be exploited by attackers to compromise the integrity, confidentiality, or availability of a system. These vulnerabilities arise from various sources, including coding errors, design flaws, misconfigurations, and inadequate security measures. Understanding and addressing these vulnerabilities is fundamental to maintaining robust cybersecurity and protecting systems from malicious threats. In general, vulnerabilities can affect any component of a software system, including operating systems, applications, network protocols, and hardware. They often provide an entry point for attackers to execute unauthorized actions, such as gaining unauthorized access to data, escalating privileges, or disrupting system operations. The impact of a vulnerability ranges from minor inconveniences to severe breaches that result in significant financial loss or reputation damage. Multiple stakeholders are involved in the discovery and management of vulnerabilities, including software developers, security researchers, and system administrators. Effective management typically includes identifying vulnerabilities through various methods such as static and dynamic analysis, penetration testing, and monitoring for reported issues. Once identified, vulnerabilities need to be assessed and addressed using appropriate mitigation strategies to alleviate their severity and potential impact.

2.3.1 Vulnerability lifecycle

The lifecycle of a security vulnerability encompasses several distinct phases that reflect the progression from discovery to resolution. Understanding these phases is crucial for effective vulnerability management and for implementing appropriate countermeasures to mitigate potential risks.

The **Black Risk** phase begins when a vulnerability is initially discovered but has not yet been publicly disclosed. During this period, the vulnerability is known only to a selected group of individuals, such as the discoverer or potentially malicious actors who might exploit it. Identifying the exact moment when a vulnerability enters the Black Risk phase is often challenging. As described by [10], this phase starts the first date a vulnerability is described on an information channel where the disclosed information on the vulnerability. In this phase, the focus is on understanding the vulnerability's potential impact and assessing risks while keeping the information confidential to prevent exploitation. The main objective is to prevent the vulnerability from being exploited until it can be properly addressed.

The **Gray Risk** phase starts with the public disclosure of the vulnerability. At this point, details about the vulnerability are made available through security advisories, public databases like the NVD, or other channels. The disclosure of the vulnerability allows for the development and distribution of mitigation strategies, such as security patches or workarounds. Although public disclosure increases awareness and drives the development of solutions, it also means that attackers have access to information that could be used to exploit the vulnerability before systems can be adequately protected.

Finally, the **White Risk** phase follows the disclosure and involves the period after a vulnerability has been publicly known and mitigations have been implemented. During this phase, the vulnerability is well-documented, and countermeasures such as security patches or updates are in place to address the issue. The immediate threat posed by the vulnerability is significantly reduced as both the software producers and users are aware of the issue and have the means to protect against it. However, ongoing vigilance is necessary to ensure that all affected systems are updated and to monitor for any residual threats or new vulnerabilities that may emerge.

Continuous monitoring and CTI play a vital role throughout these phases. CTI involves gathering and analyzing information about potential vulnerabilities and emerging threats. It helps organizations stay informed about new vulnerabilities and threats, facilitating timely responses and proactive security measures [11, 30]. By understanding and effectively managing the lifecycle of vulnerabilities, organizations can enhance their cybersecurity posture and better prepare for and respond to potential threats.

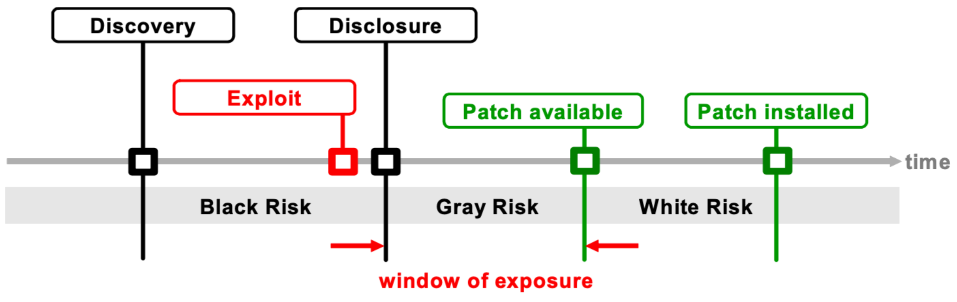


Figure 2.6: Lifecycle of a vulnerability [10]

2.3.2 CVE Identifiers

Common Vulnerabilities and Exposures (CVE) identifiers are a crucial component in the field of cybersecurity, providing a standardized method for identifying and cataloging vulnerabilities and exposures in software systems. This system facilitates the exchange of information about vulnerabilities and helps ensure that different organizations and tools refer to the same issues consistently. A CVE identifier is a unique, standardized reference assigned to a specific vulnerability or exposure. The CVE system, managed by the MITRE Corporation, assigns these identifiers to vulnerabilities based on a well-defined process[31]. Each CVE identifier includes a unique number, a brief description of the vulnerability, and relevant metadata such as the affected products, the severity of the issue, and potential impacts.

2.3.2.1 Process of Assigning CVE Identifiers

The process for assigning CVE identifiers involves several steps[32]:

1. **Discovery:** A vulnerability is discovered and reported either by security researchers, vendors, or other stakeholders.
2. **Request Submission:** A request for an identifier is submitted.
3. **Evaluation:** The CVE Numbering Authorities (CNA) evaluates the request and ensures that the vulnerability is unique and meets the criteria for a CVE identifier[31].
4. **Assignment:** Once approved, a CVE identifier is assigned and publishes the associated information in the database.

2.4 Related work

2.4.1 Related work on vulnerability detection via social networks

In recent years, the frequency and sophistication of cyber attacks have significantly increased, while organizations face ever-shorter time frames to respond effectively. To address these evolving threats, organizations have improved their vulnerability management processes, and increased the internal dissemination of security information. Vulnerability information is also disseminated through social media platforms and other informal channels [33]. It is suggested that vulnerabilities may be discussed on Twitter prior to their official public disclosure [34]. Despite these observations, there is a lack of empirical research verifying this assumption. Some studies addresses this gap by extracting Tweets that contain vulnerability-related information and evaluating whether Twitter offers a timelier source of vulnerability information[34, 35].

2.4.1.1 The Tweet Advantage

To investigate this hypothesis, the paper ”The Tweet Advantage” [35] analyzed tweets containing CVE identifiers within a specific timeframe. As mentioned earlier, CVE identifiers are often assigned before official public disclosure on formal channels. The study collected 709,880 tweets between May 23, 2016, and March 27, 2018, and mapped these tweets to the vulnerability lifecycle model. The analysis revealed that a significant portion of vulnerabilities were discussed on Twitter before their public disclosure by official entities or vendors. Specifically, one quarter of the vulnerabilities examined were mentioned on Twitter before being officially announced. The study demonstrated that Twitter can provide a valuable time advantage for reacting to newly discovered vulnerabilities. Additionally, Twitter serves as a platform for security crowdsourcing, where information reaches a wide audience of users and organizations.

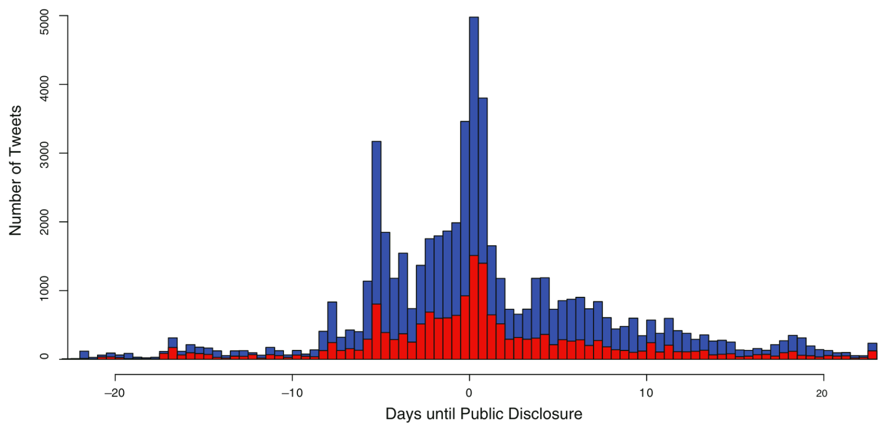


Figure 2.7: Tweet-Disclosure Pattern. The Y-axis displays the number of tweets, while the X-axis represents the timeline in days, with 0 indicating the time of public disclosure. Red bars indicate the total number of tweets (excluding retweets), and blue bars show the count of retweets[35].

Chapter 3

Methodology

This chapter outlines the methodology employed in this research. Section 3.1 provides an overview of the research process and the execution of the project. Section 3.2 focuses on the data collection process, detailing the specific choices made in designing and implementing the data pipelines. Section 3.3 describes the classification methodologies used to analyze and categorize GitHub issues.

3.1 Research Process

This research is fundamentally grounded in a rigorous process of inquiry, exploration, and validation of experiments related to the potential of LLM technology. This cutting-edge technology is currently attracting significant interest and research efforts but the objective of detecting vulnerabilities using data collected from sources such as blogs, mailing lists, and similar platforms is relatively novel and remains largely unexplored. Given the considerable flexibility in designing the system, various structures were evaluated throughout the development process. Since this approach is both innovative and uncharted, a thorough review of the literature was crucial to establish a strong foundation. Building on this, creating a dataset that met our specific requirements was essential. This dataset was designed to correlate vulnerabilities effectively with GitHub issues found in open-source code. Our work introduces a novel method for identifying vulnerabilities through the analysis of GitHub issues, additionally, this research aims to showcase the capabilities of LLMs in extracting and interpreting contextual information.

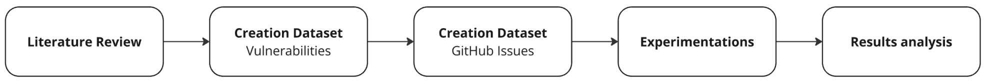

Literature Review
Creation Dataset Vulnerabilities
Creation Dataset GitHub Issues
Experimentations
Results analysis

Figure 3.1: Methodology process

3.2 Data Collection

This chapter is dedicated to the process of acquiring the data used for experimentation. As mentioned above, a classification approach was applied to GitHub issues based on their relevance to identifying vulnerabilities. Given that this approach involves supervised learning, it was essential to label each issue in the dataset to indicate its relevance to the vulnerability identification. To achieve this, information from the NVD was utilized. This database is crucial as it establishes a correlation between the vulnerabilities and the issues created at the time of problem discovery.

3.2.1 The National Vulnerability Database (NVD)

The National Vulnerability Database (NVD) is a comprehensive resource for information on known vulnerabilities in software and hardware systems. Maintained by National Institute of Standards and Technology (NIST), the NVD provides a standardized platform for identifying and tracking vulnerabilities through CVE system. NVD serves as a central repository for vulnerability data, offering detailed descriptions, severity ratings, and impact assessments. Each entry in the database includes critical information such as CVE-ID, affected software versions, and potential impact. Additionally, NVD provides links to further resources, such as issue tracking, released notes, and vendor advisories. The database is updated regularly to include new vulnerabilities as they are discovered and reported. NVD is widely used by cybersecurity professionals, researchers, and organizations to stay informed about potential security threats and to aid in vulnerability management processes. With the help of NVD, users can access reliable and up-to-date information to enhance their security measures and responses.

3.2.2 GitHub Issues

GitHub Issues is a feature within the GitHub platform that allows users to track and manage tasks, bugs, and feature requests related to software projects. It provides a collaborative environment where developers and contributors can report problems, suggest improvements, and discuss solutions. Each issue on GitHub is a record of a specific problem or request related to a repository. Users can also comment on issues, attach files, and link related issues or pull requests. GitHub Issues plays a crucial role in the open-source development process by serving as a communication tool between developers and the community. It enables users to provide feedback, report vulnerabilities, and contribute to the development and improvement of software projects. As a result, GitHub Issues serves as a valuable source of real-time information on software issues, including security-related concerns, which can be used for vulnerability detection and analysis.

3.2.3 Data selection

A critical aspect of this work was the identification and appropriate handling of the dataset used for vulnerability recognition. Given that the proposed approach is novel and lacks precedent in the literature, various formats and methods were experimented with to tackle the problem. The chosen methodology involves a model designed to classify whether an issue is relevant for identifying a vulnerability.

The dataset collection was divided into several pipelines that transformed data from sources such as the NVD and GitHub into a final dataset suitable for classification. Throughout this process, several decisions and assumptions were made to ensure that the dataset was balanced and flexible. This approach aimed to enhance the effectiveness and adaptability of the dataset for the classification task.

3.2.3.1 NVD pipeline

This initial pipeline aims to extract all information related to vulnerabilities published between January 1, 2019, and June 2, 2024. The extracted data includes details such as CVE-ID, description of the vulnerability, evaluation metrics, and the reference list. The impact metrics used to define the severity of a vulnerability are generated by the Common Vulnerability Scoring System (CVSS). Specifically, this research focuses on the Impact Score, which describes the potential effects an exploit of the vulnerability may cause at worst.

Vulnerabilities under examination or rejected were excluded from the dataset. After the pipeline, we collect a total of 113,735 vulnerabilities. For each remaining vulnerability, the list of references was extracted and used to create a consolidated dataset that correlates these references with their respective CVE-IDs. The distribution of reference types is illustrated in Table 3.1. Meanwhile, Table 3.2 shows the frequency of domains used in these references. From this data, we can infer that GitHub and Issue Tracking are integral resources for defining and understanding vulnerabilities.

Reference Type	**Count**
Third Party Advisory	86,801
Vendor Advisory	39,607
Exploit	35,246
Patch	32,283
Issue Tracking	12,364
VDB Entry	12,019
Mailing List	8,091
Permissions Required	6,198
Release Notes	5,646
Product	3,905
Broken Link	2,858
Mitigation	1,697
US Government Resource	1,003

Table 3.1: Distribution of Reference Types in the Vulnerabilities Dataset

Domain	**Count**
github.com	42,165
git.kernel.org	10,461
vuldb.com	8,332
lists.fedoraproject.org	4,790
patchstack.com	4,256
lists.apache.org	3,805
www.zerodayinitiative.com	3,620
packetstormsecurity.com	3,523
portal.msrc.microsoft.com	3,340
plugins.trac.wordpress.org	3,258
wpscan.com	3,238
exchange.xforce.ibmcloud.com	3,048
www.wordfence.com	2,752
www.openwall.com	2,700
bugzilla.redhat.com	2,531

Table 3.2: Top Frequent Domains Used in References

3.2.3.2 GitHub pipeline

Throughout the references, a total of 6,626 GitHub repositories were cited within the time frame under examination. To manage the dataset effectively, it was necessary to narrow down the number of repositories. To achieve this, the top 50 repositories with the highest number of related vulnerabilities in the last year were selected. This selection aims to retain the maximum number of repositories within the dataset that are associated with tracking issues.

Owner	**Repository**	**References**
gpac	gpac	260
axiomatic-systems	Bento4	93
jerryscript-project	jerryscript	91
ImageMagick	ImageMagick	76
LibreDWG	libredwg	70
strukturag	libde265	52
kubernetes	kubernetes	43
openlink	virtuoso-opensource	33
libming	libming	29
Piwigo	Piwigo	24

Table 3.3: Top 10 GitHub Repositories cited in the CVE references from January 1, 2019, to June 2, 2024.

In each selected repository, the goal is to identify tracking issues cited in the vulnerability references to use them as targets for classification. Additionally, other issues from the same repository will be proportionally included to enrich the classifier with non-vulnerability-related issues.

The final step in this phase involved filtering the dataset by removing repositories that did not contain a sufficient number of issues and eliminating any issues that exceeded 8,191 tokens. This token limit corresponds to the context window size of one of the classification models used.

3.3 Classification methodologies

Since this is a novel task for models like LLMs, a variety of classification methods were explored to address the challenge of classifying issues. In this chapter, we will delve into three proposed solutions that utilize different types of models, including embedding and encoding techniques. Each of these approaches leverages advanced methodologies to enhance the effectiveness of classification in identifying vulnerabilities.

The first solution employs embedding models, which transform issues into dense vectors in a high-dimensional space, capturing semantic similarities between them. The second approach utilizes encoding models that process sequences of tokens to encode contextual information, thereby improving the model's ability to understand the nuances of each issue. Lastly, we will explore a hybrid method that combines elements of both embedding and encoding strategies to achieve a more comprehensive classification.

3.3.1 Classification Using Embedding Models

Embedding models convert text into dense, high-dimensional vectors that capture semantic relationships and contextual nuances. These models represent each issue as a vector in a continuous space, allowing us to identify similarities and patterns that might not be evident in the raw text. Embedding techniques are particularly useful in capturing the underlying meaning of issues by placing similar issues closer together in the vector space. This representation helps in discerning the relevance of issues to vulnerabilities by analyzing the geometric relationships between vectors. For the embedding generation, we utilized the `text-embedding-3-large` model from OpenAI[29].

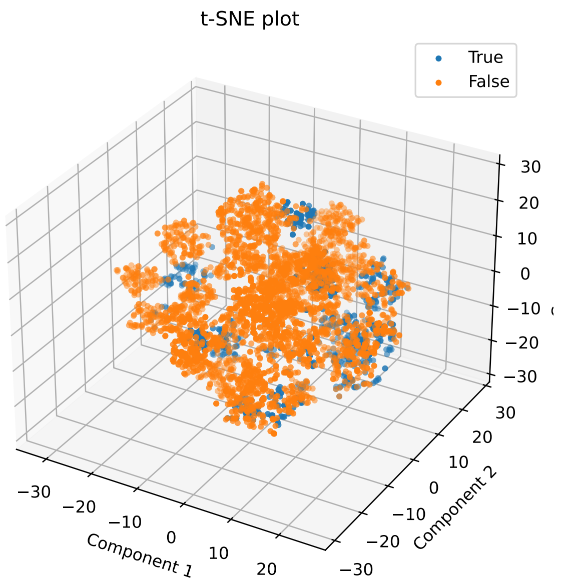


Figure 3.2: t-SNE plot of the dataset showing the distribution of issues. Relevant issues are marked in blue, while non-relevant ones are in orange.

Figure 3.2 illustrates the distribution of our dataset, revealing how issues cluster within the embedding space and indicating the potential for effective classification.

For the classification task, the XGBoost model was employed[36]. XGBoost is a robust gradient-boosting algorithm renowned for its high performance in classification tasks. Its selection was based on its ability to handle large datasets, its resilience to overfitting, and its efficiency in generating accurate predictions. By using the embeddings as input features, the model predicts whether an issue is relevant to a vulnerability, thereby enhancing the classification process through the utilization of the structured information derived from the embedding models.

The prompt used for feeding data into the embedding model is as follows:

Prompt template: display issue

This is a GitHub Issue
repo:{repo_name}
owner:{repo_owner}
Title : {title}
— start of the body —
{body}
— end of the body —

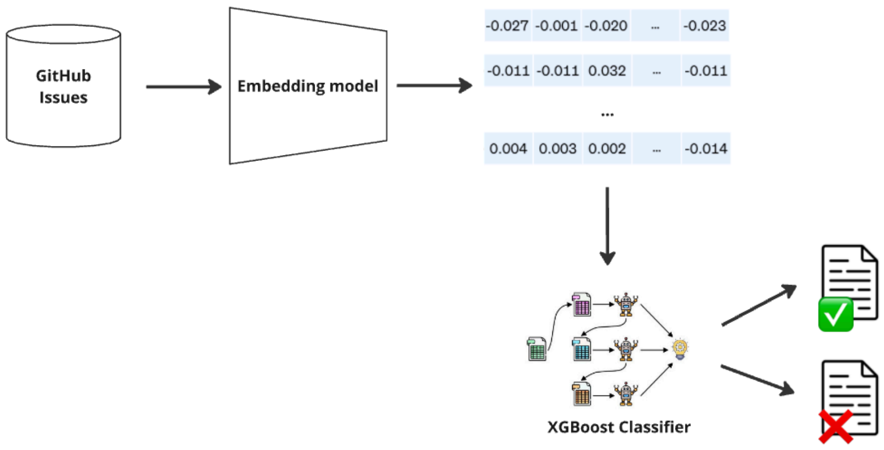


Figure 3.3: Embedding classifier

3.3.2 LLMs-based classifiers

In this methodology, LLMs are employed to determine whether a GitHub issue is relevant for identifying a vulnerability. The utilization of LLMs offers several advantages, such as a potentially more nuanced analysis of the provided context and the capability to request the model to provide a description of the problem along with a confidence score. This technique of prompt engineering, which involves asking the model to reason through the problem before providing an answer, is known as Chain-of-Thought (CoT) prompting [37]. CoT prompting enables complex reasoning capabilities by facilitating intermediate reasoning steps.

For this approach, the model is prompted to provide the following information in a JSON format:

- `gpt_description`: Describe the detected vulnerability, or write `None` if no vulnerability is detected.
- `gpt_confidence`: An integer from 1 to 5 indicating the level of confidence in the detection (1 = very low, 5 = very high).
- `gpt_is_relevant`: A boolean indicating whether the issue is relevant (`true`) or not (`false`). An issue is considered relevant only when `gpt_confidence` is 5.

By using this approach, the model generates a detailed description and a confidence score for each issue, which helps in evaluating the relevance and accuracy of the classification process. The generated description is then compared with the official description of the NVD vulnerability itself. This comparison helps assessing whether the model has accurately detected and described the vulnerability, providing an additional layer of verification for the classification process.

System prompt

You are a cybersecurity assistant tasked with identifying potential vulnerabilities by analyzing GitHub issues. Your goal is to review each issue and determine whether it indicates a security vulnerability. If you are confident that a vulnerability exists, provide a detailed description of the issue and mark it as relevant. To minimize false positives, carefully analyze the context of the message to assess if the issue is communicating a vulnerability.

In addition to identifying security vulnerabilities, you should also recognize cases where the issue is not a vulnerability. These may include failing tests, minor bugs, or issues related to functionality that do not present security risks.

Please format your response in JSON with the following fields:

gpt_description: Describe the vulnerability detected, if any. If no vulnerability is detected, write None.

gpt_confidence: An integer from 1 to 5 indicating your level of confidence in the detection (1 = very low, 2 = low, 3 = medium, 4 = high, 5 = very high).

gpt_is_relevant: A boolean indicating whether the issue is relevant (true) or not (false). An issue is considered relevant only when gpt_confidence is 5.

3.3.2.1 Zero-shot learning classification

Zero-shot learning leverages pre-trained models' ability to generalize from their broad training data to new, unseen tasks[38]. This approach relies on the model's inherent knowledge and its capability to infer the relevance of an issue based on the textual description provided.

In this method, the model is given the system prompt that outlines the task and a message containing the issue under examination.

3.3.2.2 Few-Shot learning classification

Few-shot learning leverages the capacity of pre-trained models to adapt to new tasks with only a small number of annotated examples. This approach builds on the model's existing knowledge and enhances its performance by providing specific examples that illustrate the task at hand. In this method, a system prompt is designed to outline the classification task, accompanied by a few representative examples. These examples consist of issues labeled with their relevance to vulnerabilities, which guide the model in understanding the classification requirements. The descriptions of vulnerabilities used in the examples are derived from NVD, which helps the model learn the stylistic and descriptive conventions used in vulnerability descriptions.

3.3.3 Combined classification approach

The final methodology combines both embedding and LLMs techniques to optimize vulnerability detection. This hybrid approach aims to provide a more robust and efficient classification system. Initially, issues are processed using embedding models, this step allows for a preliminary filtering of issues, grouping similar issues together and identifying those that may be relevant to vulnerabilities based on their vector representations. Following this initial filtering, the refined issues undergo a more detailed analysis using LLMs. By combining these approaches, the methodology effectively reduces the computational load on the LLMs, as only a subset of the issues filtered by the embedding classifier resulting in a more cost-effective and efficient classification process.

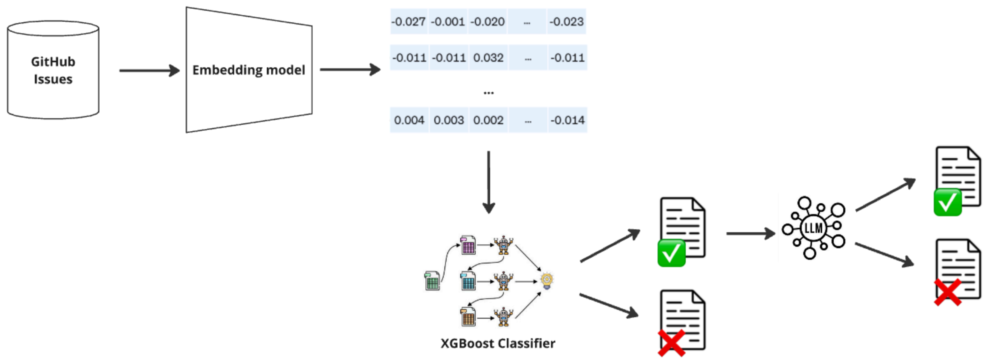


Figure 3.4: Combined solution

3.4 Performance Metrics

In this study, the evaluation of the classification models was conducted using standard performance metrics commonly employed in machine learning tasks, specifically in the context of classification problems. The metrics used include Precision, Recall, F1-Score, and overall Accuracy.

Class 1 was designated for all issues relevant to vulnerability detection, while *Class 0* was used for issues that are not relevant.

- **Precision** measures the proportion of true positive predictions among all positive predictions, reflecting the model's ability to avoid false positives. In our case, Precision indicates the proportion of issues identified as relevant that are actually relevant.

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

  where TP is the number of true positives and FP is the number of false positives.

- **Recall**, also known as sensitivity or true positive rate, measures the proportion of true positive predictions among all actual positives, indicating the model's effectiveness in capturing relevant instances. Thus, Recall measures the model's ability to correctly identify issues relevant to vulnerability detection.

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

  where FN is the number of false negatives.

- **F1-Score** is the harmonic mean of Precision and Recall, providing a single metric that balances the trade-off between them. This score is particularly useful in cases of uneven class distribution, as in our scenario.

  $$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In addition to the per-class metrics, the overall Accuracy of the model was calculated. Accuracy is defined as the proportion of correct predictions (both true positives and true negatives) among the total number of predictions, providing a general measure of the model's performance across all classes.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where TN represents the true negatives.

Chapter 4

Results and Analysis

This chapter provides an in-depth analysis of the results obtained from applying the various classification methodologies introduced earlier for evaluating GitHub issues in terms of their relevance to vulnerability discovery. The analysis will focus on several key aspects: the performance metrics of binary classification, including accuracy, precision, recall, and F1 score; the computational cost associated with each method; and the quality of vulnerability descriptions generated by approaches utilizing Large Language Models LLMs. By examining these elements, the chapter aims to offer a comprehensive evaluation of the effectiveness, efficiency, and practical implications of the different classification strategies.

4.1 Baseline Approach

To establish a benchmark for comparison with the limited existing implementations in the literature, a baseline approach was developed based on techniques described in the paper ”The Tweet Advantage”[35]. This approach utilizes statistical methods focused on identifying specific keywords that are indicative of vulnerability descriptions. The original method involved the use of regular expressions (regex) to detect CVE-IDs within the text. For this implementation, additional keyword detection was incorporated, including terms such as ”vulnerability”, ”NVD” and ”security” to enhance the identification of relevant issues.

Although the reported accuracy of 0.83 might appear to be high, as shown in 4.1, it is important to recognize that this dataset is significantly imbalanced,

Table 4.1: Detailed classification metrics for baseline approach

Class	Classification Metrics			
	Precision	Recall	F1-Score	Support
0	0.85	0.96	0.90	1 414
1	0.64	0.26	0.37	338
Accuracy		0.83		

with a predominant number of negative class instances, hence not relevant issues. Therefore, the overall accuracy metric might not fully reflect the model's performance, especially regarding the minority class.

To more accurately assess the effectiveness of the baseline approach, it is crucial to focus on the metrics related to the positive class (class 1). The precision, recall, and F1-score for the True class indicate how well the model identifies relevant instances of vulnerabilities. Specifically, the recall of 0.26 for the True class suggests that the baseline model has limited success in identifying positive instances.

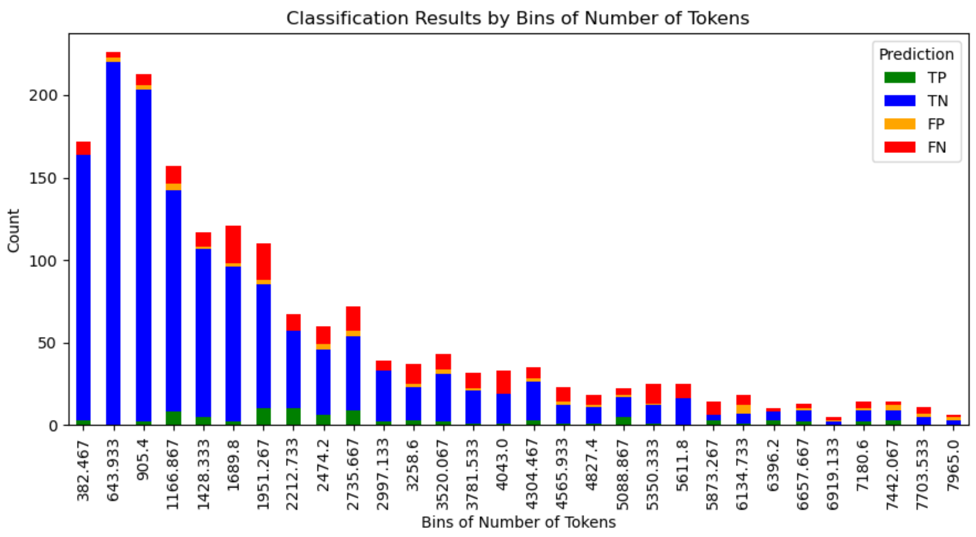


Figure 4.1: Classification performance across different issue sizes, measured in tokens and divided into 30 bins using the tiktoken tokenizer.

4.2 LLM-Based Classifiers

We now turn to the first novel approach adopted for this task. As previously outlined, two distinct techniques were employed: zero-shot learning and few-shot learning. These models, leveraging their text interpretation capabilities, were used to assess each issue's relevance to the discovery of vulnerabilities. For each issue, the models delivered a confidence assessment regarding its relevance to vulnerability discovery. When the confidence was high, the model also generated a detailed description of the associated vulnerability. This process not only provided a binary relevance assessment but also offered a useful explanation of the nature of the vulnerability, if present.

LLM utilized for this task is the ”GPT-4o-mini” from OpenAI. Due to its speed and powerful capabilities at a relatively low cost of use, it is well-suited for analyzing large volumes of data. These characteristics make it an ideal model for handling extensive datasets and performing in-depth vulnerability assessments.

Model	Class	Precision	Recall	F1-Score	Support	Accuracy
Zero-Shot	0	1.00	0.72	0.84	1 414	0.78
	1	0.46	0.99	0.63	338	0.78
Few-Shot	0	0.99	0.75	0.85	1 414	0.79
	1	0.48	0.98	0.65	338	0.79

Table 4.2: Detailed Classification Metrics for Zero-Shot and Few-Shot Models

Both the zero-shot and few-shot learning approaches demonstrate distinct strengths and weaknesses. The zero-shot model exhibits high precision for non-relevant issues, effectively minimizing false positives in this category. However, it faces challenges with false positives for relevant issues, indicating a trade-off between precision and recall. The few-shot learning model, while still exhibiting a notable rate of false positives, shows an improvement in precision for relevant issues compared to the zero-shot model. This approach also achieves a better overall F1-score, indicating a more balanced performance between precision and recall. Despite this improvement, both models display a high recall, suggesting a tendency towards flagging issues as relevant more frequently than may be accurate. During the evaluation phase, numerous prompts were tested in an effort to mitigate this bias towards positive predictions. Despite these adjustments, the general tendency of both models remains towards positive classification.

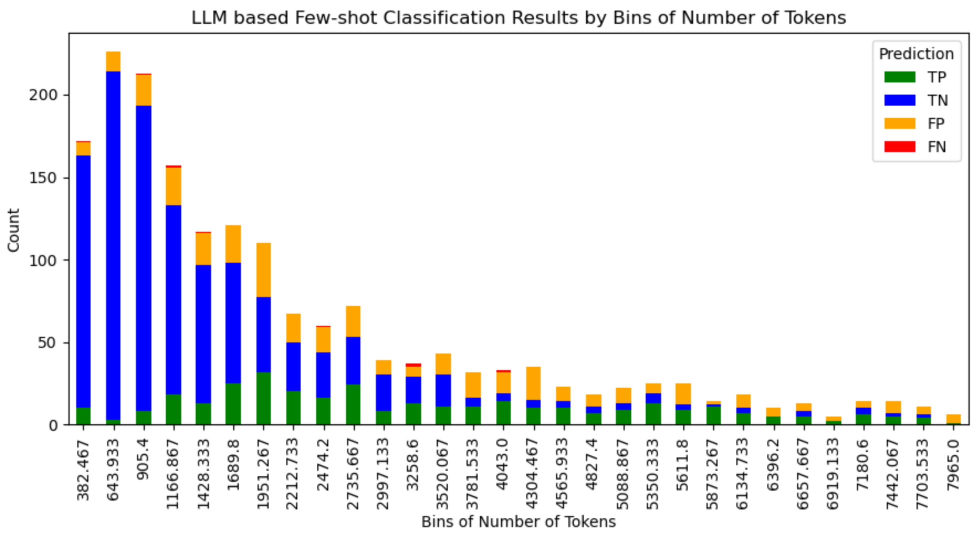


Figure 4.2: Classification performance of LLM-based model with few-shot learningacross different issue sizes, measured in tokens and divided into 30 bins using the tiktoken tokenizer.

Model	TP	TN	FP	FN	Total
Zero-Shot	333	1 025	389	5	1 752
Few-Shot	330	1 061	353	8	1 752

Table 4.3: Counts of True Positives, True Negatives, False Positives, and False Negatives for Zero-Shot and Few-Shot Models

4.2.1 Descriptions similarity

As previously mentioned, one of the key advantages of utilizing LLM is their ability to generate and elaborate on textual content. In our experiment, we leveraged this capability by requesting the model to provide a description of the identified vulnerability. This was intended to prompt the model to reason through the classification of each issue more thoroughly. By comparing the

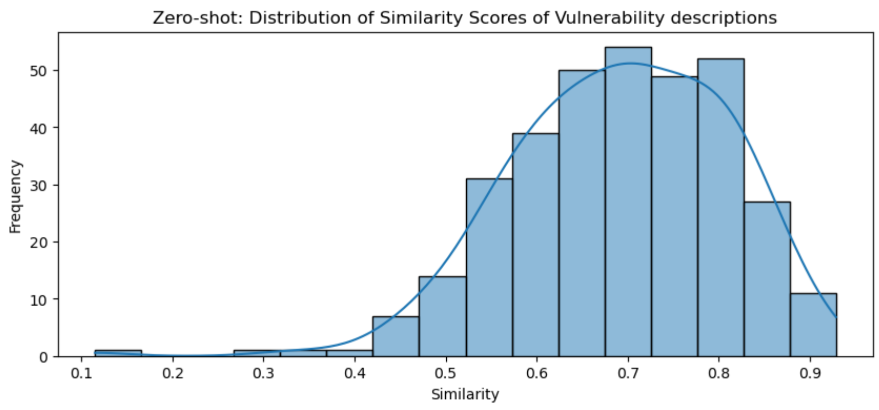


Figure 4.3: Zero-shot: Similarity distribution of the description generated for correct classified vulnerabilities

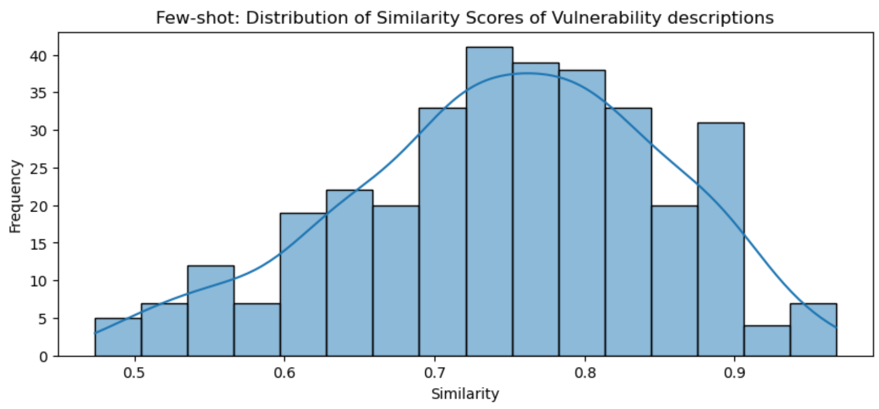


Figure 4.4: Few-shot: Similarity distribution of the description generated for correct classified vulnerabilities

Metric	Zero-Shot	Few-Shot	Delta %
Mean Similarity	0.691	0.746	7.8 %
Median Similarity	0.695	0.753	8.3 %

Table 4.4: Similarity Scores and Improvement Delta for Zero-Shot and Few-Shot Models

generated descriptions with those available in the NVD, we aimed to assess the accuracy and relevance of the model's classifications.

As illustrated in Figures 4.3 and 4.9, using examples to improve the description of vulnerabilities has achieved the desired effect. With an average similarity score of 7.8%, the model that incorporated example descriptions effectively learned the style and format of those found in the NVD.

4.3 Embedding based Classifiers

We now turn to analyzing the results obtained from the approach that leverages embeddings and XGBoost for accurate GitHub issue classification. For generating embeddings, we employed OpenAI's "text-embedding-3-large" model. This model supports a maximum context window of 8191 tokens. Consequently, during the dataset preparation phase, issues exceeding this token limit were excluded.

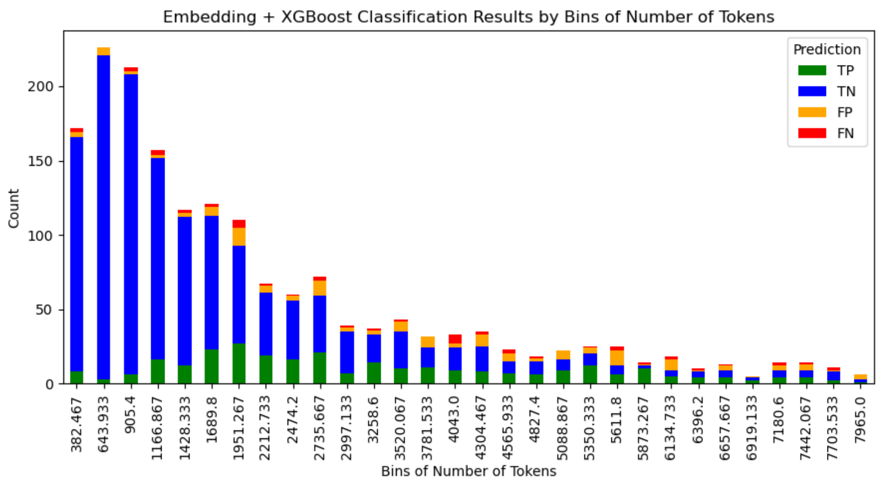


Figure 4.5: Classification performance of Embedding based classifier across different issue sizes, measured in tokens and divided into 30 bins using the tiktoken tokenizer.

Since the embedding model is also based on transformer architecture, the model focus on different parts of the input sequence, effectively capturing contextual relationships and semantic nuances. By representing each issue as a high-dimensional vector, the embeddings facilitate the detection of intricate patterns and similarities that are crucial for effective classification. The combination of these embeddings with XGBoost for classification provided a robust framework for analyzing the relevance of GitHub issues to vulnerability detection.

Table 4.5: Detailed classification metrics for the embedding based approach

	Classification Metrics			
Class	Precision	Recall	F1-Score	Support
0	0.96	0.91	0.93	1 414
1	0.68	0.85	0.75	338
Accuracy		0.89		

As shown in Table 4.5, the results obtained with this approach are very promising. The model demonstrates a good balance, and the precision for the positive class indicates a favorable trade-off between True Positives and False Positives. Additionally, this method proves to be highly efficient from a computational standpoint since the execution cost for the embedding model is an order of magnitude lower compared to that of LLM. This efficiency makes it a practical choice for large-scale applications like this task. It is also worth noting that the XGBoost classifier was optimized using the hyperparameter ”scale_pos_weight.” This parameter adjusts the weight of positive samples during training, which helps to address class imbalance by giving more importance to the minority class. By fine-tuning this hyperparameter, the model’s performance on the positive class was improved, further enhancing the overall classification accuracy.

As shown in Figure 4.6, the model exhibits a high Area Under the Curve (AUC) for the Receiver Operating Characteristic (ROC) curve. This indicates that our model performs exceptionally well in distinguishing between relevant and non-relevant issues. A high AUC value signifies that the model is capable of effectively ranking positive instances higher than negative ones, which is crucial for accurate classification in imbalanced datasets. Additionally, the Precision-Recall (PR) curve in the same figure demonstrates strong performance.

4.4 Combined Classification Approach

In this hybrid approach, an embedding-based classifier was used as a filter for the Large Language Model (LLM) classifier. This method leverages the advantages of both techniques. The embedding-based classifier provides a

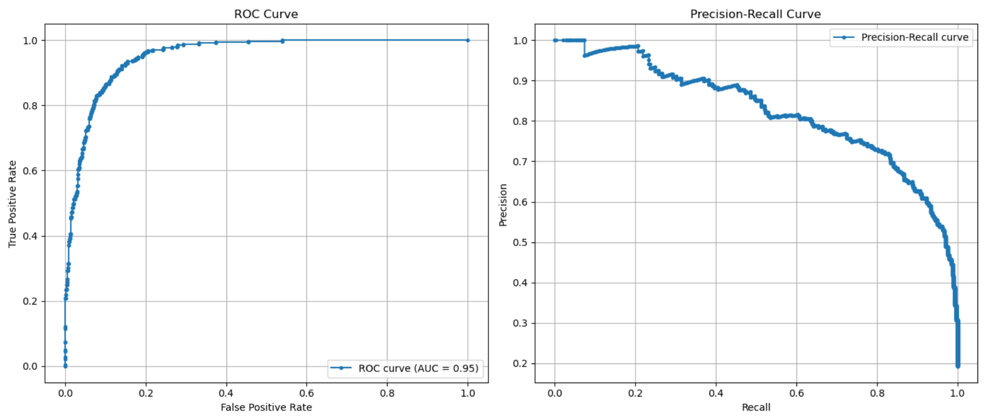


Figure 4.6: ROC and Precision-Recall curves for the XGBoost model. The ROC curve illustrates the trade-off between the true positive rate and the false positive rate. The Precision-Recall curve shows the relationship between precision and recall.

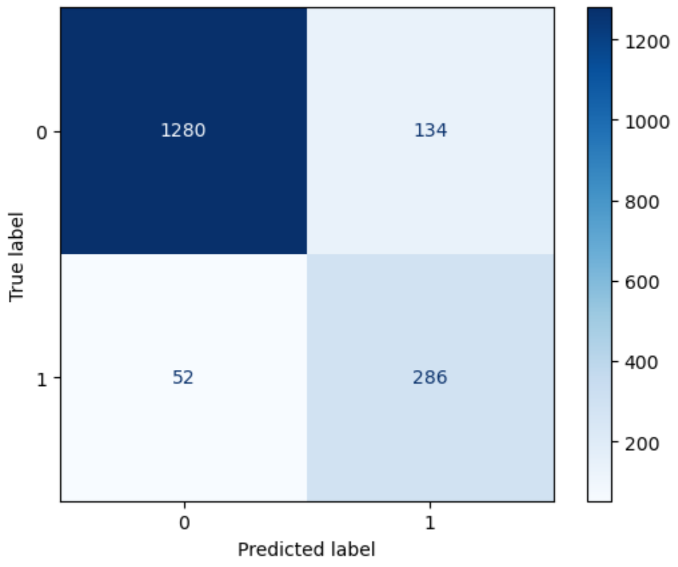


Figure 4.7: Confusion matrix showing the performance of the XGBoost classifier. The matrix displays the TP, FP, TN, and FN counts.

balanced classification, while the LLM classifier offers a detailed reanalysis of the issues along with a corresponding description of the identified vulnerabilities. By combining these methods, we achieve a more efficient classification process that balances accuracy with the ability to generate insightful descriptions of potential vulnerabilities.

Table 4.6: Detailed classification metrics for the updated approach

	Classification Metrics			
Class	Precision	Recall	F1-Score	Support
0	0.96	0.91	0.93	1 414
1	0.69	0.84	0.76	338
Accuracy		0.90		

Although the enhancement in classification performance compared to the embedding-only model is relatively modest, the integration of the LLM classifier with few-shot learning provides a more detailed analysis of the issues. The LLM's detailed reanalysis enhances the overall results by offering richer insights into each issue. This combination effectively leverages the strengths of both techniques.

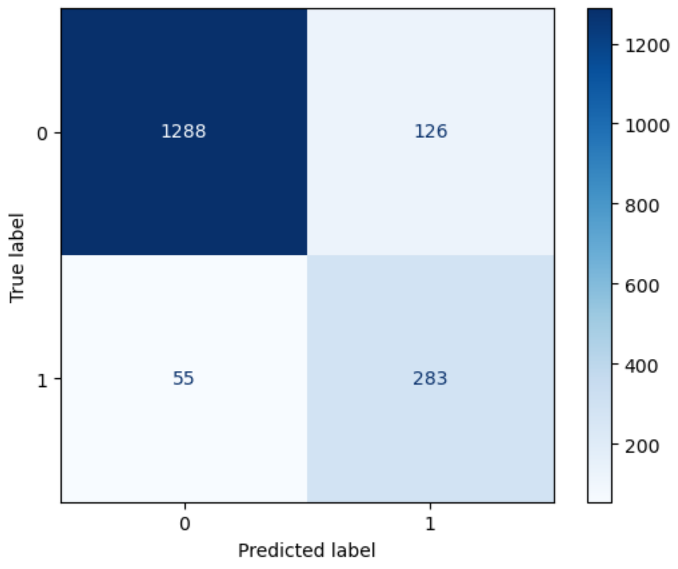


Figure 4.8: Confusion matrix showing the performance of hybrid classifier. The matrix displays the TP, FP, TN, and FN counts.

4.5 General comparison

In this section, we compare the various methodologies applied for classifying GitHub issues with respect to their effectiveness in identifying vulnerabilities. The methodologies under review include the baseline approach, zero-shot learning, few-shot learning, XGBoost classifier and combined approach.

Model	Class	Precision	Recall	F1-Score	Accuracy
Baseline	0	0.85	**0.96**	0.90	0.83
	1	0.64	0.26	0.37	
LLM: Zero-Shot	0	1.00	0.72	0.84	0.78
	1	0.46	**0.99**	0.63	
LLM: Few-Shot	0	0.99	0.75	0.85	0.79
	1	0.48	0.98	0.65	
Emb + XGB	0	**0.96**	0.91	**0.93**	0.89
	1	0.68	**0.85**	0.75	
Combined	0	**0.96**	0.91	**0.93**	**0.90**
	1	**0.69**	0.84	**0.76**	

Table 4.7: Summary of the classification performances of all the methodologies

When comparing all methodologies, the hybrid approach emerges as the most effective, combining high classification accuracy with detailed vulnerability descriptions. While the baseline approach provides a useful benchmark, both zero-shot and few-shot learning methods demonstrate improvements in relevance assessment, with few-shot learning performs better. The hybrid model benefits from the strengths of the embedding technique, which supports the approach, and integrates it with LLM reanalysis to offer a comprehensive solution for vulnerability detection and description.

4.6 Model Performance on Post-Knowledge Cutoff Vulnerabilities

In this analysis, we evaluate, to the extent possible, the performance of our model on the portion of the dataset that includes issues and vulnerabilities that arose after the knowledge cutoff date of the GPT-4o-mini model, specifically after October 2023[39]. As discussed in detail in the chapter on *Threats to Validity*, we lack visibility into the composition of the training set used for the LLM. Therefore, the only viable method to ensure that our solutions do not suffer from overfitting is to evaluate the model using data that is post-cutoff.

As explained in the following chapter, it was not feasible to use such a dataset from the outset because the model employed is relatively new, and there are few vulnerabilities that have been reported after this cutoff date. Consequently, while our evaluation does include a subset of data that fits these criteria, the available dataset is quite limited.

Specifically, within our test set, there are only 29 tracking issues that are post-cutoff and 672 normal issues. Therefore, the results obtained from this analysis should be considered partial due to the reduced size of the dataset used. Nonetheless, this evaluation provides critical insights into the model's ability to generalize to new, unseen vulnerabilities, offering a preliminary indication of its robustness and applicability in real-world scenarios beyond its training data.

	Classification Metrics			
Class	Precision	Recall	F1-Score	Support
0	0.99	0.95	0.97	672
1	0.41	0.83	0.55	29
Accuracy		0.94		

Table 4.8: Detailed classification metrics for combined methodology on post knowledge cutoff test set

The support of the new dataset, defined as the number of occurrences of the two classes within the dataset, has significantly changed compared to the initial dataset. Consequently, the proportion of Class 1 has shifted from 19% to 4%. This change results in a lower percentage of relevant issues within the new sample, aligning more closely with a realistic scenario in a potential real-world application. However, this disparity between the distributions of the two datasets complicates direct comparison of the results.

4.6.1 Descriptions similarity

It is also crucial to examine the quality and relevance of the vulnerability descriptions generated by the LLM for post-knowledge cutoff issues. This feature is particularly valuable for understanding and addressing newly emerging threats that were not part of the model's training data.

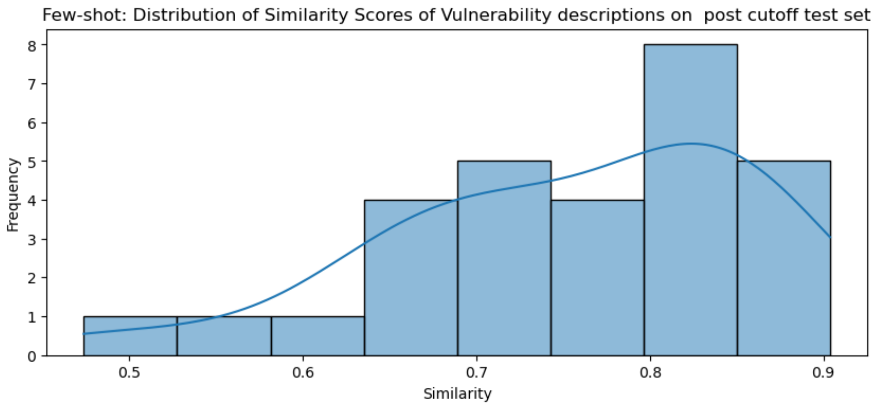


Figure 4.9: Distribution of similarity scores for descriptions generated by the few-shot model on the post-knowledge cutoff test set

Metric	Few-Shot
Mean Similarity	0.756
Median Similarity	0.775

Table 4.9: Similarity Scores for the Few-Shot Model

The similarity scores across different datasets suggest that the model effectively generalizes to new, unseen issues, reinforcing the reliability of the model. This consistency is promising and implies that the model's performance is robust and not limited to the initial training set.

Chapter 5

Discussion

5.1 Challenges

Applying transformer-based models to this problem is relatively new, making the identification of the most effective approach a primary challenge. In a world rich with diverse information exchange channels, concentrating on a single source was essential for this study. GitHub was chosen due to its extensive use and the depth of its issue tracking system, which, as shown by analyses, is frequently referenced for vulnerabilities.

Processing large volumes of data with models that have high computational costs necessitated the development of an efficient data filtering pipeline to ensure the practicality of the solution in real-world applications. By utilizing embedding models, we successfully minimized reliance on computationally expensive LLM.

Another challenge encountered was selecting an appropriate dataset for issue classification. It is important to note that the approach employed relies on a subset of potentially relevant issues for vulnerability detection. Despite this limitation, given the aim of creating a dataset with a substantial number of entries through an automated process, this was the best compromise.

Furthermore, as this is an innovative approach to the problem, there are several areas where performance improvements could be made. This includes exploring more advanced classification models beyond XGBoost and refining prompt engineering for LLM. Nonetheless, the primary goal of this research is to demonstrate the feasibility and applicability of this approach for detecting vulnerabilities using GitHub issues.

5.2 Threats to Validity

An important factor to consider regarding the validity of the results is the training of the embedding models and LLMs. These models are trained on large amounts of data sourced from the internet. Since these models are developed by private companies that have not disclosed detailed information about the composition of their training datasets, it is difficult to determine whether correlations between issues and vulnerabilities were already known to the model.

This concern was taken into account from the outset of this research. To mitigate this issue, the methodology employed involved using data and information that were created after the models' knowledge cutoff date—the date corresponding to the most recent data used for their training. However, we encountered a limitation: for the latest generation LLMs we intended to use in this experiment, there were not enough vulnerabilities with corresponding references created post-cutoff to constitute a robust test set.

Another important factor to consider is that, given this research's focus on the analysis of a large quantity of data, we did not manually verify each of the issues examined or those used as ground truth. As a result, we must assume the presence of some noise in our classifications, and it cannot be guaranteed that the issues considered non-relevant are indeed irrelevant to undiscovered vulnerabilities or that they have not already been associated with a vulnerability on the NVD.

5.3 Future Work

As discussed in the challenges of this project, the primary objective of this research was to demonstrate the effectiveness of the proposed approaches, rather than to fully optimize the models for maximum performance. For this reason, there is great room for improvement.

5.3.1 Exploration of Alternative Models

In this study, we limited our experimentation to models provided by OpenAI. However, to obtain a broader perspective on performance and capabilities, future research should explore and compare models from various providers and architectures. Evaluating models from different sources will provide insights into their relative strengths and weaknesses, and could lead to the identification of models that are better suited to this specific task. Additionally, considering the rapid advancement in the field of transformer-based models, newer models may offer enhanced performance and efficiency, making it worthwhile to continually revisit the choice of models as the technology evolves.

5.3.2 Enhancements to Embedding-Based Approaches

Significant improvements can be made to the embedding-based approach to further enhance performance. One of the most promising avenues is the fine-tuning of an embedding model specifically for this task. By doing so, the generated embeddings would be tailored to this particular application, likely resulting in more accurate classifications. Moreover, fine-tuning could be extended to integrate the classification model directly with the embedding model. Research has shown that this approach can lead to more efficient training and better overall results.

Additionally, the classification mechanism for the embeddings can be refined. Experimenting with other classification techniques, such as Support Vector Machine (SVM) and DNN, could yield improvements in accuracy and robustness, providing a more versatile and effective solution.

5.3.3 Testing on Post-Cutoff Data

To further increase the reliability of the results, future work should include testing on a dataset that is posterior to the knowledge cutoff date of the models. This would ensure that the data used for evaluation is not present in the training set, thereby providing a more accurate measure of the model's true generalization capability. Such a strategy would also help validate the robustness of the model in real-world scenarios where data evolves over time.

In summary, while this research has laid the groundwork by demonstrating the viability of the proposed approaches, there is significant scope for further optimization and validation, which could lead to even more powerful and reliable models.

Chapter 6

Conclusions

The primary objective of this project was to assess the feasibility of identifying vulnerabilities through the analysis of GitHub issues using advanced machine learning models, particularly transformer-based models. The results of this study have demonstrated that it is indeed possible to detect vulnerabilities with relatively low computational cost, especially when employing embedding models. This finding is particularly significant for real-world applications, where resource efficiency is crucial.

The research shows that leveraging embedding models can lead to accurate and timely identification of potential security issues, offering a valuable tool for developers and security professionals. By automating the detection process and incorporating advanced models like those based on embeddings and LLM, organizations can enhance their ability to preemptively identify vulnerabilities before they are disclosed publicly. This capability is vital in reducing the window of opportunity for attackers to exploit these vulnerabilities, thereby enhancing overall system security.

Moreover, the hybrid approach proposed in this work, which combines the strengths of both embedding models and LLM, has shown that it is possible to balance classification accuracy with detailed vulnerability descriptions. This integrated method not only improves the reliability of the classification process but also provides richer contextual information, which is essential for understanding the nature of the vulnerabilities detected.

Despite these promising results, the study also highlights several areas for future work, including the exploration of alternative models, the fine-tuning of embeddings specifically for this task, and the application of different classification techniques. Additionally, validating the models on data that is posterior to their knowledge cutoff is crucial for ensuring their effectiveness in dynamic, real-world environments.

In conclusion, this project has successfully demonstrated the potential of using advanced machine learning models to detect vulnerabilities through GitHub issues. The findings suggest that with further refinement and optimization, these models can be powerful tools in the ongoing effort to secure software systems against emerging threats.

References

[1] "A review of zero-day in-the-wild exploits in 2023," Mar. 2024. [Online]. Available: https://blog.google/technology/safety-security/a-review-of-zero-day-in-the-wild-exploits-in-2023/ [Page 1.]

[2] P. Törnberg, "How to use LLMs for Text Analysis," Jul. 2023, arXiv:2307.13106 [cs]. [Online]. Available: http://arxiv.org/abs/2307.13106 [Page 2.]

[3] A. Petukhova, J. P. Matos-Carvalho, and N. Fachada, "Text clustering with LLM embeddings," Mar. 2024, arXiv:2403.15112 [cs]. [Online]. Available: http://arxiv.org/abs/2403.15112 [Page 2.]

[4] X. Sun, L. Dong, X. Li, Z. Wan, S. Wang, T. Zhang, J. Li, F. Cheng, L. Lyu, F. Wu, and G. Wang, "Pushing the Limits of ChatGPT on NLP Tasks," Oct. 2023, arXiv:2306.09719 [cs]. [Online]. Available: http://arxiv.org/abs/2306.09719 [Page 2.]

[5] R. Gozalo-Brizuela and E. C. Garrido-Merchán, "A survey of Generative AI Applications," Jun. 2023, arXiv:2306.02781 [cs]. [Online]. Available: http://arxiv.org/abs/2306.02781 [Page 2.]

[6] Y. Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, "How Effective Are Neural Networks for Fixing Security Vulnerabilities," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Seattle WA USA: ACM, Jul. 2023. doi: 10.1145/3597926.3598135. ISBN 9798400702211 pp. 1282–1294. [Online]. Available: https://dl.acm.org/doi/10.1145/3597926.3598135 [Page 2.]

[7] D. Noever, "Can Large Language Models Find And Fix Vulnerable Software?" Aug. 2023, arXiv:2308.10345 [cs]. [Online]. Available: http://arxiv.org/abs/2308.10345 [Page 2.]

[8] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining Zero-Shot Vulnerability Repair with Large Language Models," Aug. 2022, arXiv:2112.02125 [cs]. [Online]. Available: http://arxiv.org/abs/2112.02125 [Page 2.]

[9] J. McHugh, W. Fithen, and W. Arbaugh, "Windows of vulnerability: a case study analysis," *Computer*, vol. 33, no. 12, pp. 52–59, Dec. 2000. doi: 10.1109/2.889093. [Online]. Available: http://ieeexplore.ieee.org/document/889093/ [Page 2.]

[10] S. Frei, B. Tellenbach, and B. Plattner, "0-Day Patch Exposing Vendors (In)security Performance," 2008. [Pages ix, 2, 14, and 15.]

[11] "IBM Security X-Force Threat Intelligence Index 2024." [Online]. Available: https://www.ibm.com/reports/threat-intelligence [Pages 3 and 15.]

[12] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits." [Page 3.]

[13] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, no. 3, pp. 3713–3744, Jan. 2023. doi: 10.1007/s11042-022-13428-4. [Online]. Available: https://link.springer.com/10.1007/s11042-022-13428-4 [Page 7.]

[14] Y. Bengio, R. Ducharme, and P. Vincent, "A Neural Probabilistic Language Model," in *Advances in Neural Information Processing Systems*, vol. 13. MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2000/hash/728f206c2a01bf572b5940d7d9a8fa4c-Abstract.html [Page 8.]

[15] R. Collobert and J. Weston, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning." [Pages 8 and 9.]

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," Dec. 2014, arXiv:1409.3215 [cs]. [Online]. Available: http://arxiv.org/abs/1409.3215 [Page 8.]

[17] "Hierarchical Transformers for User Semantic Similarity - ICWE 2023," Jun. 2023. [Online]. Available: https://www.slideshare.net/slideshow/h

ierarchical-transformersfor-user-semantic-similarity-icwe-2023/258328252 [Pages ix and 8.]

[18] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” May 2016, arXiv:1409.0473 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1409.0473 [Pages 8 and 9.]

[19] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, “A Structured Self-attentive Sentence Embedding,” Mar. 2017, arXiv:1703.03130 [cs]. [Online]. Available: http://arxiv.org/abs/1703.03130 [Pages ix and 9.]

[20] Z. Niu, G. Zhong, and H. Yu, “A review on the attention mechanism of deep learning,” *Neurocomputing*, vol. 452, pp. 48–62, Sep. 2021. doi: 10.1016/j.neucom.2021.03.091. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122100477X [Page 9.]

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . u. Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html [Pages ix, 9, 11, 12, and 13.]

[22] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, “Massive Exploration of Neural Machine Translation Architectures,” Mar. 2017, arXiv:1703.03906 [cs]. [Online]. Available: http://arxiv.org/abs/1703.03906 [Page 9.]

[23] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov,

Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open Foundation and Fine-Tuned Chat Models,” Jul. 2023, arXiv:2307.09288 [cs]. [Online]. Available: http://arxiv.org/abs/2307.09288 [Page 10.]

[24] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. doi: 10.1038/nature14539 Publisher: Nature Publishing Group. [Online]. Available: https://hal.science/hal-04206682 [Page 10.]

[25] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. doi: 10.1016/j.neunet.2014.09.003 ArXiv:1404.7828 [cs]. [Online]. Available: http://arxiv.org/abs/1404.7828 [Page 10.]

[26] malcolmp, “A Non-Technical Introduction to Transformers,” Jun. 2023. [Online]. Available: https://medium.com/healthcare-ai-sig/a-non-technical-introduction-to-transformers-6ca36b8246d [Pages ix and 11.]

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” May 2019, arXiv:1810.04805 [cs]. [Online]. Available: http://arxiv.org/abs/1810.04805 [Page 13.]

[28] S. Alazmi and D. C. De Leon, “A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners,” *IEEE Access*, vol. 10, pp. 33 200–33 219, 2022. doi: 10.1109/ACCESS.2022.3161522. [Online]. Available: https://ieeexplore.ieee.org/document/9739725/ [Page 13.]

[29] “New embedding models and API updates.” [Online]. Available: https://openai.com/index/new-embedding-models-and-api-updates/ [Pages ix, 13, and 24.]

[30] T. D. Wagner, K. Mahbub, E. Palomar, and A. E. Abdallah, “Cyber threat intelligence sharing: Survey and research directions,” *Computers & Security*, vol. 87, p. 101589, Nov. 2019. doi: 10.1016/j.cose.2019.101589. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016740481830467X [Page 15.]

[31] J. Lin, B. Adams, and A. E. Hassan, “On the coordination of vulnerability fixes,” *Empirical Software Engineering*, vol. 28, no. 6, p.

151, Nov. 2023. doi: 10.1007/s10664-023-10403-x. [Online]. Available: https://doi.org/10.1007/s10664-023-10403-x [Page 16.]

[32] "Process | CVE." [Online]. Available: https://www.cve.org/About/Process [Page 16.]

[33] C. Sauerwein, C. Sillaber, and R. Breu, *Shadow Cyber Threat Intelligence and Its Use in Information Security and Risk Management Processes*, Mar. 2018. [Page 16.]

[34] R. Syed, M. Rahafrooz, and J. M. Keisler, "What it takes to get retweeted: An analysis of software vulnerability messages," *Computers in Human Behavior*, vol. 80, pp. 207–215, Mar. 2018. doi: 10.1016/j.chb.2017.11.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747563217306623 [Page 16.]

[35] C. Sauerwein, C. Sillaber, M. M. Huber, A. Mussmann, and R. Breu, "The Tweet Advantage: An Empirical Analysis of 0-Day Vulnerability Information Shared on Twitter," in *ICT Systems Security and Privacy Protection*, L. J. Janczewski and M. Kutyłowski, Eds. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-99828-$2_15. ISBN 978 - 3 - 319 - 99828 - 2 pp. 201 - -215. [Pages\ ix, 16, 17, and\ 31.]$

[36] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016. doi: 10.1145/2939672.2939785 pp. 785–794, arXiv:1603.02754 [cs]. [Online]. Available: http://arxiv.org/abs/1603.02754 [Page 25.]

[37] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Jan. 2023, arXiv:2201.11903 [cs]. [Online]. Available: http://arxiv.org/abs/2201.11903 [Page 26.]

[38] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, X.-Z. Wang, and Q. M. J. Wu, "A Review of Generalized Zero-Shot Learning Methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2022. doi: 10.1109/TPAMI.2022.3191696 ArXiv:2011.08641 [cs]. [Online]. Available: http://arxiv.org/abs/2011.08641 [Page 27.]

[39] "OpenAI Platform." [Online]. Available: https://platform.openai.com [Page 41.]