Hong Kong International Journal of Research Studies, ISSN: 3078-4018
Volume 2, Issue 2, July December, 2024
Available online at: https://octopuspublication.com/index.php/hkijrs

Using Large Language Models to Discover and Manage Cybersecurity Threats and Software Vulnerabilities

Geetha Priya D T[1], Dr. Rajesh Koolwal[2]

[1]Research Scholar, Department of Computer Science, JS University, Shikohabad, Uttar Pradesh
[2]Assistant Professor& Supervisor, Department of Computer Science, JS University, Shikohabad, Uttar Pradesh

ABSTRACT

The rapid growth of digital systems has significantly increased the complexity and frequency of cybersecurity threats and software vulnerabilities. Traditional security mechanisms often struggle to detect sophisticated, evolving attacks in a timely manner.

Recent advances in Large Language Models (LLMs) offer new opportunities to enhance cybersecurity by enabling intelligent analysis of large volumes of unstructured data, including logs, vulnerability reports, threat intelligence feeds, and source code. This study explores the application of LLMs in discovering, analyzing, and managing cybersecurity threats and software vulnerabilities.

The research highlights how LLMs can support tasks such as automated vulnerability detection, threat classification, incident response assistance, and security knowledge extraction through natural language understanding and contextual reasoning. By leveraging their ability to identify patterns, correlate diverse data sources, and generate actionable insights, LLMs improve the accuracy and efficiency of cybersecurity operations.

The study also discusses challenges related to model reliability, data privacy, and adversarial misuse. Overall, the findings suggest that Large Language Models have strong potential to augment existing cybersecurity frameworks, enabling proactive threat detection and more effective vulnerability management in modern software systems.

INTRODUCTION

Data corruption, performance drops, or even exploitable security flaws might be signs of an issue with the program's execution. This highlights the importance of promptly identifying and fixing mistakes. Both more traditional methods, such as static analysis, and more modern ones, based on deep learning, have their limits when it comes to effectively discovering software flaws. Despite their usefulness in finding vulnerabilities early on, static analysis and DL approaches aren't without their flaws, such as an interpretability difficulty and feature selection issues that lead to a high false positive rate. When these false positives happen, developer productivity could be severely affected.

A lot of people in the software engineering field are curious in the new developments in deep learning (DL) and how they could solve old problems. A lot of people have suggested fixing software problems using Deep Learning (DL) methods. Notable DL-based solutions to these problems include DeepRepair [49] and DLFix [48]. Using a deep learning model trained on a corpus of source code, CURE [50] is able to understand and replicate the code structure. Using a hybrid technique that combines deep learning with spectrum-based fault location, DEAR [51] enhances code context awareness. To find the best ways for DL-based program defect eradication, further study is needed to compare and contrast each of these approaches.

Yet, Natural Language Processing (NLP) has been rocked by the emergence of Large Language Models (LLMs) driven by the groundbreaking Trans-formers architecture. This has ushered in a new age of capabilities and applications, leading to tremendous improvements. Recent studies have shown that LLMs outperform deep learning (DL) and other traditional approaches when it comes to finding security vulnerabilities. [9]

LITERATURE REVIEW

Software bug reports are among the most important components of software. A variety of tracking systems, such as Jira, Bugzilla, Trac, and others, ensure that they are always up to date. Important information on the severity, importance, and remedy is provided by them if a software feature fails to function properly. In addition to that, it includes specifics of the input that was supplied by a number of programmers in order to fix the problem. The purpose of this study is to explore

software libraries via the use of text analytics. In this study, a number of different text mining strategies are investigated. These techniques include data extraction, categorization, retrieval, and summarization from issues that have been reported with software. All of the information that is obtained from software bug reports is arranged in a manner that makes it simple to locate the many aspects that are essential. Text mining and machine learning are two methods that are used in the process of developing models for bug prediction.

Ontology creation is used to generate models that are able to extract information from the text of software problem reports. The material is then summarized once the models have been constructed. At the end of the chapter, a summary of the findings that were obtained by searching through a huge body of literature on software flaws using a variety of text mining approaches is presented.

METHODOLOGY

Bug reports, source code, change logs, and a plethora of other software assets are all painstakingly documented inside the Issue Tracking System. There has been a lot of research on these subjects throughout the years. Though laborious and time-consuming, the process of deleting these software packages from issue tracking repositories is a good one in the end. Data related to software issues is often disorganized, making it hard to draw actionable conclusions. Automated data extraction experts have been trying to figure out a method to skip this tedious process. In order to glean structural details from issue reports, N. Bettenburg and colleagues developed InfoZilla.

The report's revision history, stack traces, and source code are all part of these components. There was an assumption that including these components would make machine learning more effective when training it. Stack traces, enumeration filters, and patch filters were all used throughout this procedure. By reviewing the bug reports sent to Eclipse, we were able to gauge the tool's performance [37].

M. Nayrolles and colleagues postulated the possibility of a method to get information from bug tracking and version control databases. The acronym "BUMPER" refers for "Bug Metarepository for Developers and Researchers," and it describes their method. It is an open-source, free-to-use web app that gathers information from user inquiries. Based on the amount of sophistication they need, inquiries may be categorized as either basic or sophisticated. This bug reporting program is compatible with five separate platforms: Gnome, Eclipse, Netbeans, and the Apache Software Foundation. The biggest drawback of this tool is that it requires a deep knowledge of query language to obtain data from it [38]. There is a method to get bug reports, as shown by Y. Yuk and colleagues.

The suggested procedure begins with collecting raw data from several sources, then crawls that data, and then transforms it into a data tree structure. In order to get the appropriate symbols for each issue report, XML parsers traversed these data trees [39]. R. Malhotra and colleagues developed the Configuration Management System (CMS) to provide an alternate to conventional technologies that extract textual information from issue reports. This utility can determine the overall amount of errors in each category by analyzing CVS log files. It also shows how extensive the changes were and which versions of the application were changed [40]. Table 2.1 summarizes the previous attempts to locate and eliminate software artifacts.

3. GUARANTEED BUGS DETECTION

An error that drastically alters the system's behavior from its intended state is called a software bug [67]. You may be able to get information about bugs from those who report them. It is standard practice to notify the relevant parties when an issue arises with software while it is in use by filing a bug report. The date, version, issue description, severity, and priority are some of the pieces of information that are included in each bug report.

Bug reports may be examined by experts with the use of code traces and a detailed, common language description of the issue. In order to address or further analyze each issue, many programmers provide suggestions in the form of constructive remarks. Many businesses rely on bug report tracking systems to keep tabs on and resolve a broad range of issues that arise during software development projects. This simplifies software maintenance significantly.

All parties involved in the project, including developers, testers, and others, will find these tactics useful since they facilitate communication and the sharing of diverse issues. Bug reports, requests for changes, additions, extra jobs, and subtasks are just a few examples of the numerous potential types of difficulties. Here are a few of examples. There are a plethora of issue tracking systems available nowadays, including Jira, Bugzilla, Trac, and Mantis. Regardless, among the alternatives, Git and Bugzilla are the most popular and well-known. Information that might be valuable could be found in bug reports.

Hong Kong International Journal of Research Studies, ISSN: 3078-4018
Volume 2, Issue 2, July December, 2024
Available online at: https://octopuspublication.com/index.php/hkijrs

A. Bug reports as a means of data extraction

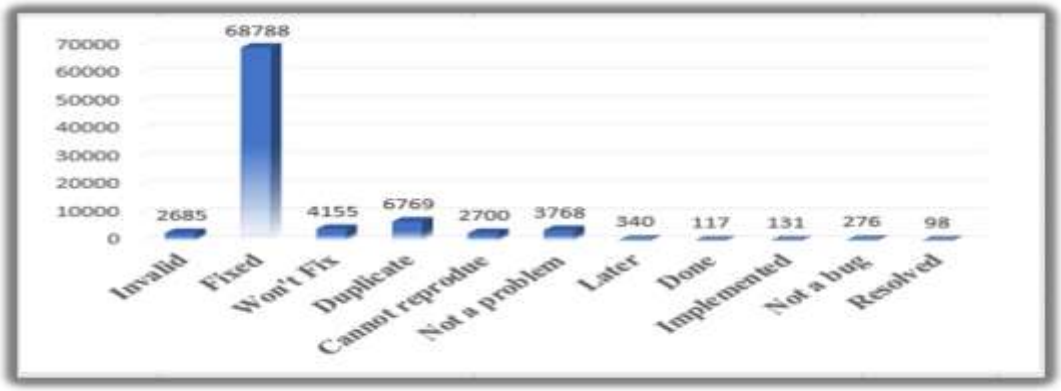


Fig. 3.4 Quantity of Issue Reports Resolved

The degree of damage that an issue causes to a software system is directly proportional to the fact that the problem is severe. The severity may be broken down into five different levels: important, blocker, not important, major, and critical. They may also be further classed as either major or non-severe, depending on the severity of the condition. problems that are less severe are referred to be minor and trivial, whereas problems that are more significant are referred to as critical, stopper, and major.

B. Finding and Fixing Software Issues

Through an examination of the textual descriptions of bug reports and the comments made by a variety of writers, this section investigates software flaws that have been found in a number of Apache Software Foundation (ASF) projects. When significant problems are not addressed and fixed, the usefulness and efficiency of a system are reduced, as stated by resolution. In order to make the system more efficient and effective, it is essential to determine the most serious problems that have not been handled and then allocate them to different members of the staff. Section 3.5.1 enumerates the most critical issues that have not yet been resolved for each and every project that is managed by the Apache Software Foundation. Section 3.5.2 identifies the employees who have made the most contributions to each project in order to resolve the most critical issues. This is done in order to accomplish the aforementioned goal.

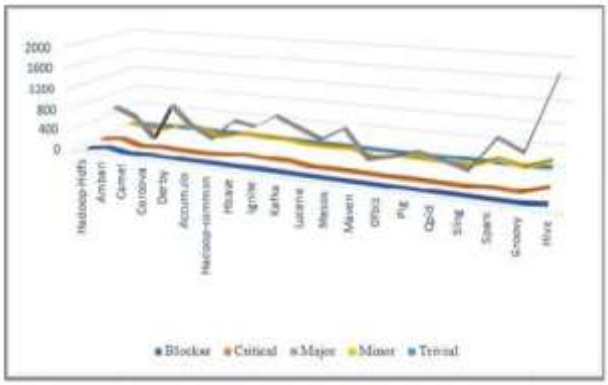


Fig. 3.6. Quantity of Unresolved Bugs with Varying Levels of Severity

C. Potentially Error-Prone Sections of Software Systems

An analysis of linguistic components, such as a To find the most common issue regions in bug reports, you need a one-line explanation, a detailed description, and a lot of notes. Machine learning serves several functions in text mining, one of which is the categorization of data. A document must undergo pre-processing procedures before it can be used to produce a feature vector for automated document sorting. Finding the words, phrases, and characters that will be used as building blocks in subsequent operations is known as pre-processing in the area of text mining. There are three steps to the process: breaking, tokenization, and stop word deletion. 1. String sequences such as words, phrases, keywords, and so on may be

"tokenized" by separating them into their component elements. All string sequences may be processed in this way. In order to remove commas and marks, change all capital letters to lowercase, and transform all capital letters to lowercase, text mining is a necessary step. Tokens, who are units of action, are formed by the use of language. The correct way to use this expression is to say "Have a good day!" while combining the words "have," "a," "good," and "day." (2) Eliminating superfluous words and phrases: What we call "stop words" are really clusters of words that appear often across all languages. Commonly used terms in literature typically have no purpose other than as adjectives, prepositions, conjunctions, or adjectivals. Here are a few examples: "and," "are," "this," "the," and "in." Their ineffectiveness as document organizers necessitates their disposal. To improve the system's performance, zero in on the most important phrases, and reduce the quantity of text data, stop words must be eliminated. The third stage, stemming, aims to isolate the most important parts of various word and grammatical forms, such as nouns, adjectives, verbs, and others like them. The goal is to simplify all of the word forms while reducing them to a single standard, including their inflated and original varieties. The suffixes -ly, -ed, -ing, -er, and others may be eliminated. Using this method while playing word search can help you remember more and feel less anxious.

Categorization of Error	Significant Keywords
Logical code error	Throws, exception, divide, unhandled, pointer, throw, uncaught, null pointer, raises, systemoutofmemoryexception, trigger, dividezero, rethrow
Input/output error	Logging, build, classpath, inputformat, api-jarfilemissing, log, loggers, imports, initialized, requestchannels, mapoutputlocationgetfile, displayed, console log
Resource Allocation error	Task_allocation, buffers, memory, synchronized, configuration, memoryfailure, runtime, dynamic, bucketcache
Network Related error	Datanode, localhost, address, port, domain, security, process, https, global, interfaces, binding, virtual, bindexcept, limits

Table 3.7. Types of Error and Important Terms Related to Them

4. PREDICTION OF BUG SEVERITY USING TEXT

Since text mining was first used to categorization, that is where much of the focus lies in this work. The term "categorization" really refers to the practice of rapidly sorting text into predefined categories. Users may report software flaws using bug tracking platforms like Jira and Bugzilla. The severity of the situation makes the software system issue, its scope, and its significance easy to see. Predicting the severity of software errors in advance has the potential to greatly enhance the process of finding and repairing software problems. The researchers have tried their best, but there have been many obstacles that have prevented them from finding the optimum way to use text mining to predict software quality.

It is common practice to compare a software issue's severity to two extremes: major cases and less significant instances. There are three ways to describe issues that are deemed highly serious: major errors, critical defects, and stopper flaws [10]. None of them seem to mind that the bugs are tiny and hard to see. The brief textual descriptions of software defect reports allow us to assess the program's quality. Researchers have used text mining and machine learning to categorize issues according to their severity in a number of ways; such examples are [10-12], [34], [42], [47-48], [51], [68-69], and [132-134]. The data came from a large amount of published articles on text mining as a tool for software quality evaluation. "Practical research" refers to studies that apply many machine learning algorithms to diverse datasets. Choice trees [64], Bayesian learners [135], neural networks [43], ensemble learners [136], support vector machines [60], and a plethora of others fall under this group of learning algorithms. We still don't know enough about machine learning algorithms to make reliable judgments on text mining software quality.

A. Software Security Risk Assessment Approach

Through the use of historical data and other prediction methods, it is possible to evaluate the severity of a recently identified software problem. Both decision-making and judgment-making are possible applications for heuristics, which are characterized by their ease of use and their usefulness. Representative heuristics are a tool that helps facilitate the process of group decision-making. The severity of occurrences that have not been reported may be judged with the use of heuristics. The "test set" is where they make informed predictions about the severity of the problems that have been reported, and the "training set" is where they analyze reports of bugs that have been identified as being troublesome or problematic.

Hong Kong International Journal of Research Studies, ISSN: 3078-4018
Volume 2, Issue 2, July December, 2024
Available online at: https://octopuspublication.com/index.php/hkijrs

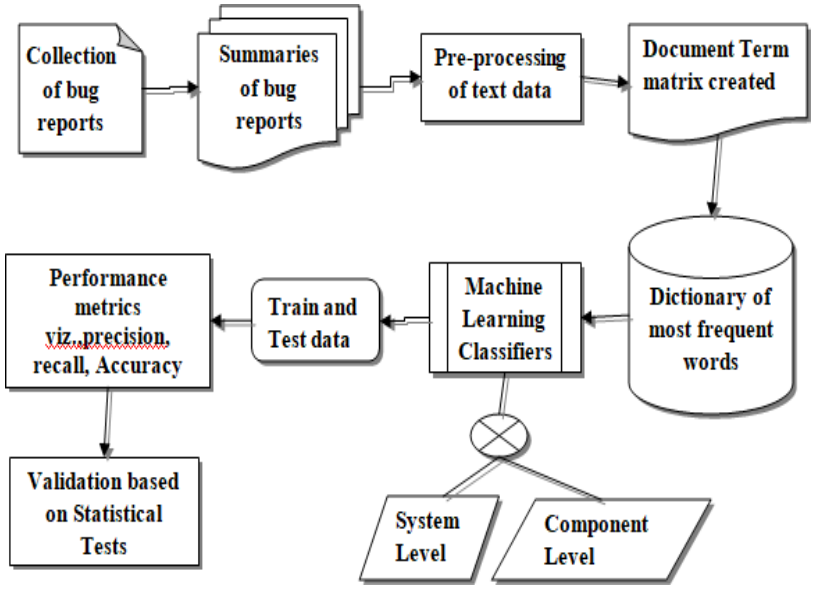


Fig. 4.1 Experimental Design of Proposed Approach

The Bug Report Collection System (BRCS) is a tool that is used for the purpose of gathering information about Apache projects. This information is given in Section 3.2 of Chapter 3. With regard to the selection of datasets, the dataset that has the greatest amount of mistakes is employed. There were thirteen datasets in the Jira folder that had the largest amount of issues. These datasets were Hadoop-HDFS, Hadoop-Common, Apache Cordova, Hive, Ambari, Groovy, Hbase, Lucene, Mesos, Maven, Qpid, Sling, and Spark. The reports that were obtained may be divided into two basic categories: those that address serious bugs and those that do not include severe bugs. It is possible to observe in Table 4.1 that each dataset has been allocated a number of concerns, some of which are significant while others are not as significant. When attempting to estimate the quality of the program at the system level, it is necessary to count the number of errors that are found in each sample. Due to the fact that the found defects are also organized by component, you are able to gain an idea of how serious the software problem is at the component level. In order to determine the extent of the problem, it is possible to choose the components that are the most faulty from among many datasets. The bar graphs that are shown in Table 4.2 and Figure 4.1(a-d) illustrate the projects and the issue regions that are associated with each of them.

Table 4.2. Software defect statistics broken down by system level severity

Project Name	#of bug reports		Total # of bug reports	Period of collection
	Non-severe	Severe		
Hadoop-hdfs	787	2622	3409	2007-2016
Hadoop-common	1021	4045	5066	2006-2016
Apache cordova	1124	3412	4536	2011-2016
Hive	730	5303	6033	2008-2016
Ambari	261	10156	10418	2011-2016
Groovy	721	3249	3970	2003-2016
Hbase	1464	4618	6082	2007-2016
Lucene	575	1321	1896	2005-2016
Mesos	253	1749	2002	2011-2016
Maven	286	1731	2017	2004-2016
Qpid	581	2665	3246	2006-2016
Sling	431	1727	2158	2007-2016
Spark	921	3323	4244	2012-2016

Hong Kong International Journal of Research Studies, ISSN: 3078-4018
Volume 2, Issue 2, July December, 2024
Available online at: https://octopuspublication.com/index.php/hkijrs

B. Data Ontology for Deductively Reasoned Software Issues

The significance of ontologies within the semantic web has grown in recent years due to their capacity to facilitate communication and data exchange. The semantic web, which Gruber created, organizes previously unstructured material so that computers can process it with little human intervention [155]. The term "a tool that retrieves ontology-based queries" may also describe a semantic search engine. Through the use of ontologies, we may establish a common understanding of subjects and develop a definition of vital knowledge. An ontology may provide a framework for defining a high-level perspective on textual meaning analysis. By using a collection of concepts and their interconnections, it offers a formal characterization of expertise in a certain field.

The word "ontology" is used to characterize this concept [156]. It is simpler to obtain data, define domains, and interact across multiple systems when ontology is combined with standard languages like Resource Description Framework (RDF) and Ontology Web Language (OWL). Even though ontology can restore our understanding of the problem, there are still areas that struggle to define ideas and their interconnections [18–20]. When applied to the domain of information, ontology allows for the organization of data and the development of a common language for data identification [157]. The reporting of software bugs is the single most critical part of software development and maintenance in the discipline of software engineering. Information such as the project name, component name, release version, brief description, and other data on software issues are included in them, along with the BugId. Several writers provide insightful criticism and comprehensive explanations on how to resolve an issue.

An great amount of work has gone into studying disorganized bug reports, with various attempts to classify complaints [42, 160], triage problems [69, 158], forecast errors [132], assess the seriousness of reports [10, 48, 53, 159], and many more. A bug report's underutilized potential includes, but is not limited to, gathering detailed technical information about the issue. That being said, compiling a roll call of everyone who contributed to fixing the Hdfs-7707 problem would be a wise move. Everyone who contributed to fixing #Hdfs-7707, not only the assigned worker, has to have their identity given. Extracting useful information from a large database of reports describing problems requires a vocabulary of terms relating to bugs. Prior studies on software bug ontologies neglected to prioritize the incorporation of problem report data, detailed descriptions, and comments into the ontology development process [21], [98], [99], and [161].

C. A New Approach to Building Ontologies

Figure 6.1 shows a diagram of the suggested procedure for creating an ontology. Reports, in-depth explanations, and comments are compiled by the Apache Bug Report Corpus (APBRC) for twenty-one different software bugs. It was developed in order to give the foundation for a theory on defects in software. The attributes are extracted and the text is pre-processed subsequent to the creation of the database. It is possible for you to conceive about BugIds as well as their properties as concepts and attributes, respectively. A formal concept grid is constructed using this method. Ontologies are formed using a variety of links, such as "is-a" and "has-a," and grids serve as the foundation for these ontologies. The study of ontology In order to make an ontology accessible on the internet, one utilizes Web Language, and in order to extract data from inquiries, one needs a reasoner.

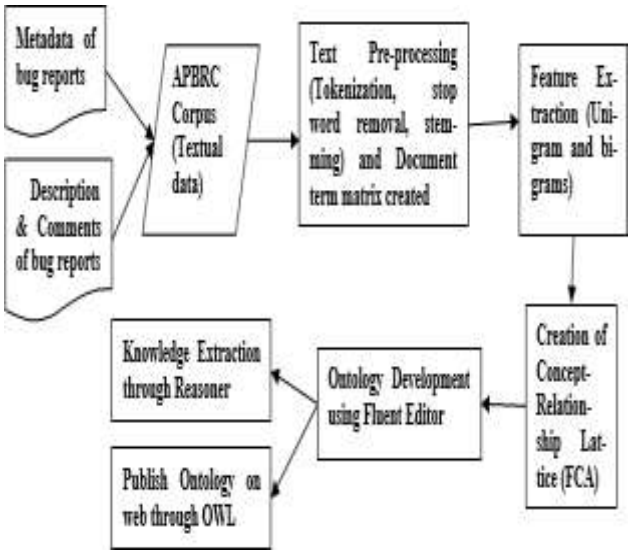


Fig 6.1 Diagram of the Ontology-Creation Process

D. Extracting Sentence and Keywords for Automatic Software Issue Summarization

In this chapter, we'll go over the second method for getting a rundown of software problem reports. A combination of lines and keywords forms the basis of this approach to automatically find software faults. A technique that leverages the logical proximity of software bug declarations was discussed in Chapter 7. The statements and lines of code that follow are not part of the package: "=", "public static," "^," "sql," "{tmp field}," "{", and "}." You could learn something from both the background information and the coding samples. Because of this, engineers are better able to comprehend the issues and devise solutions. You may show data in many different document formats using code snippets. To ensure the results are accurate and understandable, it is helpful to be acquainted with code snippets and to utilize them when reporting issues [190]. Ultimately, we hoped that this research would help us compile a comprehensive list of software defects together with applicable code samples. Bug reports are written in a way that engineers can easily comprehend and use.

CONCLUSION

Software developers can access massive amounts of unstructured data from many sources using English. Researchers are showing a growing interest in collecting and analyzing this kind of data. Text mining is a method for extracting information from unstructured datasets. Currently, researchers are looking at a wide range of text mining techniques, including methods for information extraction, sorting, retrieval, and summarization. The impact of programming mistakes is substantial across all program kinds. Consequently, several text mining techniques are used to detect software errors. In order for open-source issue systems to detect and fix software errors automatically, organized techniques are required. The primary goal of this research is to identify these techniques.

By amassing software defects from twenty projects run by the Apache Software Foundation, an Apache Project Bug Report Corpus (APBRC) is created. There is a standard structure for all software problems that includes a brief description, a response, and a severity level. Details about the error, such as its number, a detailed explanation, the responsible programmer, any feedback from other developers, and the matter's importance We utilize an automated method called the "Bug Report Collection System" (BRCS) to gather a range of characteristics.

The severity of software failures may be predicted with the help of models built using machine learning and text analytics. It is possible to classify software issues as either very serious or very minor. When you have a one-line description of a software issue that has already been addressed, you may use the Term Frequency-Inverse Document Frequency (TF-IDF) method to find the most common terms. The domain of machine learning makes use of ten methods, including boosting, Naïve Bayes, K-nearest neighbor, support vector machine, maximum entropy, decision tree, random forest, bagging, stabilized linear discriminant analysis, and generalized linear models.

We can make predictions about the system-level and component-level severity of problems. Several criteria, including as recall, precision, and accuracy, are used to evaluate the approaches. In twelve of the fourteen trials, boosting outperformed random forest, which had an accuracy range of 75% to 97%. With an accuracy range of 81% to 98%, boosting was far behind. Aside from K-Nearest Neighbor, Maximum Entropy, Decision Tree, Naïve Bayes, and Support Vector Machine all do not meet the average performance requirement. Finally, it should be mentioned that out of all the machine learning algorithms, SLDA and GLM are the two with the lowest accuracy rates. You may utilize statistical procedures like the Friedman Rank Test and the Nemenyi Post-hoc Test to make sure your findings are accurate. If there is a lot of variation among the machine learning methods, it will trigger an alarm.

REFERENCES

[1]. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J.: Haoyu Wang. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620v4 2024
[2]. Charalambous, Y., Tihanyi, N., Jain, R., Sun, Y., Ferrag, M., Amine, Cordeiro, L.: A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification. (2023). 10.48550/arXiv.2305.14752
[3]. Dipayan Saha, S., Tarek, K., Yahyaei, S.K., Saha, J., Zhou: Mark Tehranipoor, Farimah Farahmandi. LLM for SoC Security A Paradigm Shift. arXiv:2310.06046v1 2023
[4]. Ferrag, M.A., Ndhlovu, M., Tihanyi, N., Cordeiro, L.C.: Merouane Debbah, Thierry Lestable, Narinderjit Singh Thandi. Revolutionizing Cyber Threat Detection with Large Language Models. arXiv:2306.14263v2 2024.
[5]. Andreas Happe and Jürgen Cito: Getting pwn'd by AI: Penetration Testing with Large Language Models. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023). Association for Computing Machinery, New York, NY, USA, 2082–2086. (2023). https://doi.org/10.1145/3611643.3613083

[6]. Sakaoglu, S.: 'KARTAL: Web Application Vulnerability Hunting Using Large Language Models: Novel method for detecting logical vulnerabilities in web applications with finetuned Large Language Models', Dissertation, (2023)
[7]. Ferrag, M.A., Battah, A., Tihanyi, N., Debbah, M., Lestable, T.: Lucas C. Cordeiro. SecureFalcon The Next Cyber Reasoning System for Cyber Security. arXiv:2307.06616v1 2023
[8]. Weng, G., Andrzejak, A.: Automatic Bug Fixing via Deliberate Problem Solving with Large Language Models, in 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), Florence, Italy, 34–36. (2023). pp 10.1109/ISSREW60843.2023.00040
[9]. David Noever: Can Large Language Models Find and Fix Vulnerable Software. arXiv:2308.10345v1 2023
[10]. Hammond Pearce, B., Tan, B., Ahmad, R., Karri: Brendan Dolan-Gavitt. Examining Zero-Shot Vulnerability Repair with Large Language Models. arXiv:2112.02125v3 2022.
[11]. Jingxuan He and Martin Vechev: Large Language Models for Code: Security Hardening and Adversarial Testing. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23). Association for Computing Machinery, New York, NY, USA, 1865–1879. (2023).
[12]. https://doi.org/10.1145/3576915.3623175
[13]. Xia, C.S., Wei, Y.: Lingming Zhang. Practical Program Repair in the Era of Large Pre-trained Language Models. arXiv:2210.14179v1 2022
[14]. Purba, M., Ghosh, A., Radford, B., Chu, B.: Software Vulnerability Detection using Large Language Models, in 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), Florence, Italy, 112–119. (2023). pp 10.1109/ISSREW60843.2023.00058
[15]. Yuqiang, S., Wu, D., Xue, Y., Liu, H., Wang, H., Xu, Z., Xie, X., Yang Liu:. GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. arXiv:2308.03314v2 2023.
[16]. Huang, K., et al.: An Empirical Study on Fine-Tuning Large Language Models of Code for Automated Program Repair, 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, Luxembourg, pp. 1162–1174, (2023). 10.1109/ASE56229.2023.00181
[17]. Katsadouros, E., Patrikakis, C.Z., Hurlburt, G.: Can Large Language Models Better Predict Software Vulnerability? in IT Professional. May-June. 25(3), 4–8 (2023). 10.1109/MITP.2023.3284628
[18]. Marwan Omar: Detecting software vulnerabilities using Language Models. arXiv:2302.11773v1 2023
[19]. Mamede, C., Pinconschi, E., Abreu, R., Campos, J.: Exploring Transformers for Multi-Label Classification of Java Vulnerabilities, 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), Guangzhou, China, 2022, pp. 43–52, 10.1109/QRS57517.2022.00015 enchmarks. arXiv:2312.12575v2 2024