2024 32nd International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) | 979-8-3315-3130-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/MASCOTS64422.2024.10786575

Transforming the field of Vulnerability Prediction: Are Large Language Models the key?

Miltiadis Siavvas*, Ilias Kalouptsoglou*†, Erol Gelenbe‡§¶, Dionysios Kehagias*, and Dimitrios Tzovaras*
* Centre for Research and Technology Hellas/Information Technologies Institute, Thessaloniki, Greece
†University of Macedonia/Department of Applied Informatics, Thessaloniki, Greece
‡ Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (PAN), Gliwice, Poland
§ Université Côte d'Azur CNRS I3S, Nice, France
¶ King's College London, UK
siavvasm@iti.gr, iliaskaloup@iti.gr, seg@iitis.pl, diok@iti.gr, Dimitrios.Tzovaras@iti.gr

***Abstract*—Vulnerability prediction is an important mechanism for secure software development, as it enables the early identification and mitigation of software vulnerabilities. Vulnerability Prediction Models (VPMs) are Machine Learning (ML) models able to detect potentially vulnerable software components based on information retrieved from their source code. Despite the notable advancements in the field of vulnerability prediction, especially with the utilization of Deep Learning (DL) and text mining techniques, current literature still lacks a highly accurate, reliable, and practical VPM. Recently, the Large Language Models (LLMs), which have demonstrated remarkable capabilities in text understanding and processing, have started being utilized for vulnerability prediction, demonstrating highly promising results. The purpose of the present paper is to explore the utilization of LLMs in the field of vulnerability detection, identify challenges and open issues that still need to be addressed, and potentially propose directions for future research. Our analysis suggests that while LLM-based VPMs have outperformed traditional DL approaches in vulnerability prediction, significant challenges still need to be addressed to be considered sufficiently accurate, reliable, and practical.**

***Index Terms*—Software Security, Vulnerability Prediction, Large Language Models, Transformers, Review, Survey**

I. Introduction

The increasing reliance of critical daily activities on software-intensive systems, along with the high accessibility and interconnectivity of modern software, renders security an aspect of major concern. Attackers attempt to exploit vulnerabilities that reside in software in order to infringe important security policies (e.g., Confidentiality, Integrity, etc.). In fact, the exploitation of a single vulnerability can lead to devastating consequences both for the end users and for the owning enterprise of the compromised software (e.g., privacy infringement, financial and reputation damages), as revealed by the numerous examples of well-known real-world security breaches, including HeartBleed [1], WannaCry [2]. and Equifax Breach [3]. Hence, there is a strong need for techniques that enable early vulnerability identification and elimination, prior to the release of the software on the market.

One such technique is vulnerability prediction, which enables the identification of security hotspots, i.e., software components (e.g., packages, classes, functions, etc) that are likely to contain vulnerabilities. More specifically, vulnerability prediction focuses on the provision of Vulnerability Prediction Models (VPMs), which are mainly Machine Learning (ML) models able to predict the existence of vulnerabilities in software components, based on information retrieved from their source code. These models are commonly used for prioritizing testing and inspection efforts, by allocating limited test resources to potentially vulnerable parts.

A large number of VPMs have been proposed in the related literature over the years, ranging from simple ML models that utilize software metrics or simple text mining techniques (e.g., Bag of Words) for source code representation [4]–[6], to more advanced Deep Learning (DL) models, such as Long Short-Term Memory (LSTM) networks, which utilize more complex textual representation techniques (e.g., token sequences, word embeddings, text-rich graphs, etc.) [7]–[9]. Despite the advancements in the field, a VPM that is sufficiently accurate, reliable, and practical in the sense that it is capable of predicting real-world vulnerabilities in projects completely unknown to the training dataset (i.e., cross-project prediction) has not been proposed yet.

The recent advancements in the field of Natural Language Processing (NLP) with the proposition of the Transformer architecture [10] and its derivative Large Language Models (e.g., [11]–[14]), which have demonstrated remarkable human-like capabilities in text understanding and processing, has shifted the focus of the research community from traditional DL-based techniques to more advanced approaches based on Large Language Models (LLMs). LLMs, despite being originally introduced for NLP, have been examined for their utilization in various Software Engineering tasks, mainly through transfer learning, demonstrating promising results [15].

Despite their recent proposition, LLMs have already been widely examined for their capacity to be used as vulnerability predictors, showcasing highly promising results (e.g., [16], [17]). Hence, it would be of high interest to examine how the LLMs have been utilized for vulnerability prediction, how they stand as opposed to more traditional DL-based VPMs, and whether they have managed to address all the existing open issues or there are still challenges that we need to overcome.

To this end, the purpose of this paper is to review the

most notable studies in the field of LLM-based vulnerability prediction, identify existing challenges and open issues that need to be addressed, and potentially propose directions for future research. The motivation of the present work is to see, based on the existing research results, whether LLMs can be considered the "key" mechanism for achieving accurate, reliable, and practical vulnerability prediction, or more work is still required. This paper can act as a reference for researchers aiming to contribute to the vulnerability prediction field, in order to gain a solid understanding of existing solutions and identify open issues that require further research.

The rest of the paper is structured as follows: Section II provides a review of the vulnerability prediction literature putting particular emphasis on the utilization of LLMs. Section III discusses the main challenges and open issues that were identified, along with potential directions for future research. Finally, Section IV concludes the paper.

II. Literature Review

ML-based vulnerability prediction is a highly mature field with a large volume of research endeavors having been devoted to the provision of accurate vulnerability predictors. Initial attempts in the field focused mainly on the utilization of software metrics, with specific emphasis on complexity, coupling, and cohesion metrics, as potential indicators of vulnerabilities in software components, and particularly as the basic inputs for constructing early ML-based vulnerability predictors [4], [18], [19], which showcased promising results.

Subsequently, text mining approaches that represent source code in textual form for ML models to detect vulnerability patterns have been explored. Early efforts focused on simpler techniques like Bag of Words (BoW), where software components (e.g., classes, methods) are represented as token frequency vectors [5], [6], [20], [21]. These methods showed promise, outperforming metrics-based models. Later, more advanced Deep Learning (DL) techniques used word embeddings (e.g., word2vec [22], fastText [23]) and Recurrent Neural Networks (RNNs), particularly LSTM, to account for token sequences, leading to more accurate models [24].

Later, researchers started to build models based on graphical representations of the source code (i.e., text-rich graphs), such as Code Property Graphs, in order to incorporate more context from the analyzed source code into the produced predictors. To this direction, Zhou et al. proposed Devign [8], an approach that utilizes Graph Neural Networks (GNNs) to transform source code into graph representations, trying to capture the structural relationships within the code to effectively identify vulnerabilities. Then, the ReVeal study [25] demonstrated the insufficiency of current approaches to identify vulnerabilities in a realistic setting and proposed a data collection process along with a GNN-based model capable of significantly improving the vulnerability prediction process.

The above analysis indicates that the text mining-based vulnerability prediction models, especially those that were built utilizing DL (e.g., RNN and Convolutional Neural Network) have dominated the field of vulnerability prediction. This observation is also supported by a recent systematic mapping study in the field [26], which noted that the vast majority of the research work in the field of vulnerability prediction focuses on text mining and DL models. This study also noted the expected shift of the research community from simple DL approaches to the more advanced Transformer-based models.

In fact, recently, after the breakthrough in the field of NLP that was marked by the proposition of the Transformer architecture [10] and its derivative LLMs (e.g., GPT [11], [12], BERT [13], BART [14], etc.), several vulnerability prediction models have been proposed leveraging the power of the pre-trained LLMs. More specifically, a large number of research endeavors have already focused on examining various ways in which the pre-trained LLMs can be used as the basis for constructing vulnerability prediction models and comparing their effectiveness in detecting vulnerabilities to more traditional vulnerability prediction approaches.

One of the earliest attempts in the field of utilizing LLMs for vulnerability prediction [16] leveraged LLMs as a code representation technique. More specifically, the authors conducted a comparison of different Python source code representation methods for vulnerability prediction. They investigated the efficiency of word2vec, fastText, and BERT embedding vectors for code representation. In every case, they used an LSTM model to classify the embedded software components as vulnerable or not. Their findings suggested that all three techniques are suitable for representing source code for the task of vulnerability prediction, but the BERT-based embedding method was the most promising one.

Similarly to [16], in [17] the authors compared the capacity of the embedding vectors generated by CodeBERT (a variant of BERT that was pre-trained on a large corpus of software programs) to lead to more accurate vulnerability prediction models. More specifically, the authors used CodeBERT, word2vec, fastText, and GloVe as code representation techniques and utilized their produced embedding vectors to train Gated Recurrent Unit (GRU) networks for function-level vulnerability prediction. The evaluation was based on a real-world dataset containing 132,018 functions written in C, 1,983 of which were labelled as vulnerable. A synthetic dataset retrieved from the Software Assurance Reference Dataset (SARD) was also utilized for fine-tuning the CodeBERT model and for augmenting the real-world dataset in order to treat the class imbalance. The results showed that the CodeBERT-based embedding method outperformed the others in the task of vulnerability prediction, indicating the ability of the CodeBERT embedding vectors to capture more contextual information from the code potentially detecting more complex vulnerability patterns; but, with higher computational cost.

Coimbra et al. [27], evaluated the capacity of the Code2vec [28] model, which is typically used for generating embeddings for code snippets based on the structure of the source code, in predicting vulnerabilities in C functions. They also compared its predictive performance to simple Transformer-based methods, particularly RoBERTa and CodeBERT, which were fine-tuned on the Devign [8] dataset. Hence, the authors

essentially compared the graphical code representation in the form of Abstract Syntax Tree (AST) that is encapsulated in the Code2vec embeddings, with the pure textual representation of the source code that is encapsulated by simple Transformer-based models. Their findings highlight that both approaches achieved comparable results. In fact, although the performance of the Code2Vec model in vulnerability prediction was slightly lower compared to the performance of the fine-tuned versions of CodeBERT and RoBERTa, it required much fewer resources for its training rendering it a more efficient solution.

Subsequently, based on the promising results of the early research endeavors, several researchers have focused on providing dedicated LLM-based vulnerability prediction models. For instance, in [29] the authors proposed an LLM-based solution for function-level vulnerability prediction named VulBERTa, a customized version of RoBERTa [30], which is a popular BERT variant. The authors pre-trained the RoBERTa model on a large dataset of C/C++ functions (comprising approximately 2.2 million functions) from GitHub and the Draper [31] dataset in order to gain a deeper understanding of source code in general, and fine-tuned it on the downstream task of function-level vulnerability prediction based on a set of popular datasets (e.g., Reveal [25], VulDePecker [24], Devign [8], etc.). The results of the comparison of VulBERTa to other traditional RNN- and Convolutional Neural Network (CNN)-based models and more advanced and complex pre-trained LLMs (e.g., CodeBERT) revealed its effectiveness in achieving state-of-the-art results while being more resource-efficient.

In [32], VulDeBERT was introduced, which is a BERT-based vulnerability prediction model, able to provide predictions at program slice-level of granularity. VulDeBERT was built by fine-tuning the pre-trained BERT model on real-world vulnerability data retrieved from the National Vulnerability Database (NVD) and synthetic data retrieved from the SARD maintained by NIST. VulDeBERT, instead of processing directly the program slices, operates on code gadgets, which are abstract representations of the program slices typically including sequences of statements that are semantically related through data and control dependencies. VulDeBERT demonstrated higher predictive performance compared to traditional Bidirectional LSTM (BiLSTM)-based vulnerability prediction models, including the well-known VulDeePecker [24].

In [33], Ziems et al. examined whether BERT along with transfer learning from the English language can lead to accurate file-level vulnerability prediction. More specifically, the authors pre-trained BERT on a large corpus of English text retrieved from Wikipedia (2.5 billion words) and fine-tuned it for vulnerability prediction on a dataset comprising 100,000 C/C++ source code files, which were gathered manually from SARD. BERT outperformed traditional LSTM models mainly due to its attention mechanism, which better captures context within the code; however, it requires much higher computational resources to be trained and executed.

In addition, an empirical analysis was conducted in [34] by reproducing and comparing nine state-of-the-art deep learning-based vulnerability prediction models, including both traditional DL-based models (e.g., RNNs, GNNs, etc.) and modern Transformer-based models (e.g., CodeBERT, VulBERTa, etc.). The empirical evaluation was based on two C/C++ vulnerability datasets, namely Devign [8] and MSR [35]. The evaluation results showed that the transformer-based models like LineVul, CodeBERT, PLBART, VulBERTa-CNN, and VulBERTa-MLP generally perform better than the traditional DL models at line-level vulnerability prediction (i.e., detection), with LineVul (an approach based on CodeBERT) showcasing the best predictive performance in terms of F_1 score.

An empirical evaluation of four popular LLMs, including Davinci and GPT-3.5, with respect to their capacity to provide code snippet-level vulnerability prediction was conducted in [36]. The evaluation was based on two popular datasets, namely VulDeePecker [24] and CVEfixes [37] that contain vulnerable and clean (in fact, neutral) C/C++ code snippets. The results of the analysis showed that, although LLMs demonstrated a very good predictive performance in terms of recall (with Davinci being the best-performing model), their precision was found to be low (i.e., they tend to generate a significant number of false positives), and, thus, complementary utilization of program analysis techniques may be beneficial.

In addition, Zhang et al. [38] examined the capacity of LLMs in line-level vulnerability prediction, in fact, detection, considering both proprietary and open-source models. In particular, 10 popular LLMs were examined, namely GPT-3.5, GPT-4, Llama 2, CodeLlama, WizardCoder, CodeBERT, GraphCodeBERT, PLBART, CodeT5, and CodeGen. Several learning paradigms were examined including discriminative fine-tuning, generative fine-tuning, zero-shot learning, and one-shot learning. The authors, based on popular vulnerability datasets (e.g., Big-Vul [35]), created their own vulnerability datasets named BV-LOC and SC-LOC containing real-world vulnerabilities of C/C++ programs, which were used as the basis of their empirical study. The results showed that discriminative fine-tuning is the best learning approach leading to the highest possible F_1 score (demonstrated by the CodeLlama model), with some LLMs outperforming existing deep learning-based methods by over 36.2%

Zhou et al. [39] investigated the capacity of two popular LLMs, namely GPT-3 and GPT-4, in providing accurate method-level vulnerability prediction (i.e., detecting whether a given method contains vulnerabilities). The authors examined various prompt designs and compared these models with CodeBERT, which acted as a baseline model for the evaluation, focusing their evaluation on a C/C++ vulnerability dataset introduced in [40]. GPT-4 showcased the best predictive performance, outperforming the CodeBERT by 34.8% in terms of accuracy when proper prompts were used.

The LLM-based vulnerability prediction models often fail to provide more fine-grained information about the potential vulnerability (e.g., exact location) whereas the lack of transparency of the LLMs renders them difficult to understand and explain. Hence, Fu et al. [41] proposed LineVul, a Transformer-based approach for function-level vulnerability prediction capable of reporting the exact location (i.e., the

exact lines of code) in which the vulnerability resides through eXplainable Artificial Intelligence (XAI). More specifically, LineVul when it decides that a given function is potentially vulnerable, utilizes the self-attention mechanisms of its CodeBERT component, in order to identify those lines of code that influenced the most its decision, and therefore that are likely to correspond to a vulnerability pattern. Experimental results on a large-scale dataset of 188,000+ C/C++ functions demonstrated that LineVul outperforms state-of-the-art methods.

Furthermore, Sotgiu et al. [42] employed another XAI technique, namely the SHapley Additive exPlanations (SHAP) technique, in order to determine the influence of different features on the model's performance, through the identification of spurious correlations between the input features (i.e., tokens). Specifically, they fine-tuned the CodeBERT model for token-level vulnerability prediction on the Devign [8] dataset, and examined how spurious features can influence its decisions. Spurious features refer to features or patterns in the data (here tokens) that the model mistakenly learns to associate them with the target variable (e.g., class attribute), but they do not have a legitimate causal relationship with it. The results showed that CodeBERT, despite its high performance, relies heavily on special characters and irrelevant tokens, leading to concerns about its reliability, thus stressing the need for explainability-based debugging of ML-based vulnerability prediction models, for identifying and mitigating potential biases.

Similarly, the utilization of explainability techniques for localizing the vulnerabilities in the source code that are detected by DL-based function-level vulnerability predictors was investigated in [43]. More specifically, SHAP was employed on top of two DL-based models, namely VulDeePecker and JavaBERT, in order to identify vulnerability-related source code statements in Java programs. Particularly, DL models were utilized to predict the existence of vulnerabilities in functions, and SHAP was leveraged for localizing the vulnerability in the function's code. The results showed that while the models were able to accurately predict the existence of vulnerabilities in the analyzed functions, they struggled to accurately localize the exact statements that led to the models' decision, achieving only a moderate performance. The work indicates that while XAI techniques can offer some insight, the produced DL models tend to rely on features that are not genuinely relevant to the existence of vulnerabilities, indicating the need for more sophisticated methods for vulnerability localization.

As can be seen by the above analysis, although vulnerability prediction is a relatively new area of research, a large number of techniques have been examined over the years. In Table II a summary of these techniques is provided to highlight how the field evolved from simple ML-based models to the more advanced LLM-based approaches.

III. Open Challenges

Despite the fact that the Transformer-based LLMs are a very recent concept, a large volume of research endeavors have already explored the potential utilization of LLMs for vulnerability prediction. Although the LLM-based vulnerability prediction models have demonstrated much higher predictive performance compared to traditional DL-based (e.g., RNN-based and CNN-based) approaches, there are still many open issues and challenges that need to be addressed, for achieving highly accurate, reliable, and practical vulnerability prediction.

First of all, existing attempts have focused almost predominantly on evaluating the LLM-based VPMs solely on the considered datasets (i.e., within-project evaluation), omitting testing them on new and unseen data (i.e., cross-project evaluation). Demonstrating high accuracy in cross-project prediction is highly important for the LLM-based VPMs's usefulness and practicality, as this would mean that they could be used in practice without additional training or adaptation. However, some more in-depth exploratory studies have shown that their performance in cross-project prediction is generally poor [42]–[45], which can be attributed to the fact that their decisions are affected by project-specific features (i.e., tokens) as revealed by various XAI techniques. Therefore, additional work is required towards evaluating and improving the performance of LLM-based VPMs in cross-project prediction, potentially by considering more generic and project-agnostic features.

Another important open issue is the inability of existing VPMs to identify vulnerabilities that span among various source code files of a given software system. Existing VPMs are able to process sequential source code chunks of a given software component (e.g., class, function, etc.) and therefore they consider information only from the software component under analysis, omitting critical information from other parts of the broader software system. Some software vulnerabilities are highly complex and require the execution of specific code fragments of various software components in order to be triggered and successfully exploited. Hence, this may lead VPMs to miss some vulnerability patterns and to be able to detect less complex vulnerability patterns. Therefore, there is a need to propose techniques that will enable VPMs to consider more information from the broader software system, allowing them to predict the potential existence of more complex vulnerability patterns. To the best of our knowledge, the only known attempt is the recently proposed System-wide Vulnerability Assessment (SWVA) framework [46], [47].

Moreover, the frequently utilized LLMs in vulnerability prediction, such as BERT, CodeBERT, and GPT-2, are characterized by a maximum input limit of 512 tokens, not allowing the LLM-based VPMs to process lengthy software components (e.g., classes, functions, etc.) at once. This token limitation can hinder the ability of the model to fully capture the context of large components (e.g., classes), which usually exceed 512 tokens. This may lead to the omission of important vulnerabilities (i.e., generation of false negatives) or the false identification of clean code as vulnerable (i.e., generation of false positives). In addition, splitting lengthy source code into smaller pieces to fit the token limit can also disrupt the logical flow of the code, further reducing the accuracy of the model. While larger LLMs like GPT-4 and Mistral AI can handle significantly larger input sizes, they are either more costly or require substantial processing power for fine-tuning. Thus,

TABLE I
SUMMARY OF TECHNIQUES AND APPROACHES FOR VULNERABILITY PREDICTION

Technique/Approach	Main Advantages	Main Disadvantages	References
Traditional ML models using Software Metrics	Simple and easy to implement, Require low computational power	Low accuracy, May not capture actual (causal) vulnerability patterns	[4], [18], [19]
Traditional ML text mining-based models (BoW)	Easy to implement, Work well for simple vulnerability patterns	Lack semantic and syntactic understanding, Tend to miss complex vulnerability patterns	[5], [6], [19]–[21]
DL text mining-based models (sequences of tokens)	Higher accuracy than simpler ML, Capture syntactic patterns in the code	Resource-intensive, Need large datasets for training, Omit structural relationships in code	[7], [8], [24]
Graph-based models (GNNs)	Capture structural relationships in code, Better for complex patterns	Resource-intensive, Need large datasets for training, Effort to extract graph representations of code	[8], [9], [25], [27]
XAI-based techniques (e.g., SHAP, Attention)	Provide transparency and localization of vulnerabilities	Add complexity to the process, Localization performance may be moderate	[41]–[43]
Large Language Models (e.g., CodeBERT)	Best observed predictive performance, Capture deep contextual information	Resource demanding, Impose input (token) limits, Omit structural/graphical context in source code	[16], [17], [27], [29], [32]–[34], [36], [38], [39], [41]

additional research work is required to identify effective ways that would enable the LLM-based VPMs to process software components regardless of their source code size.

Furthermore, while pre-training popular LLMs on source code-specific data and fine-tuning them for vulnerability prediction led to sufficient predictive performance, the incorporation of additional context from the source code (e.g., code dependencies, execution paths, etc.) has been found to further improve accuracy [27], [36]. In particular, although models such as GraphCodeBERT, which encapsulate data flow graph information, have begun to be examined (e.g., [38]), more modalities consisting of different data-dependent features are required (e.g., data-flow graphs, control-flow graphs, code property graphs, etc.). To this end, multi-modal models capable of receiving information from text and several graphical representations should be examined. The incorporation of graphical information could potentially also improve the performance of the LLM-based VPMs in cross-project prediction (a challenge that we mentioned previously), as they are normally more abstract, containing less project-specific information as opposed to treating source code solely as text.

Another key issue is the need for a reliable, diverse, and sufficiently large vulnerability dataset. While datasets like Big-Vul [35], ReVeal [25], and CVEfixes [37] have been proposed, they exhibit weaknesses that impact the reliability of vulnerability prediction models (VPMs). These datasets often classify software components as ”vulnerable” or ”clean” based on GitHub versions, but unreported vulnerabilities in the ”clean” class lead to mislabeled data. Additionally, since fixes usually involve minor changes, the similarity between ”vulnerable” and ”clean” components makes it difficult for the produced models to distinguish between them. Thirdly, most datasets focus on specific languages (mainly C/C++) and on a limited range of software projects, reducing the diversity of the data and, in turn, the generalizability of the produced models. Therefore, there is a need for datasets containing highly diverse vulnerability information retrieved from software from various domains and programming languages, without mislabelled instances. Using Generative AI to create synthetic realistic data could be a promising solution.

Finally, another important issue is that although LLMs have outperformed other more traditional DL approaches in vulnerability prediction, it has been observed that they require much more resources and computational power especially for their pre-training and fine-tuning. This significantly hinders their practicality, as it may prevent stakeholders from using them in practice, due to the extra costs that they incure. Although prompt engineering techniques (i.e., zero-shot or few-shot learning) have been examined as a more efficient alternative to pre-training and fine-tuning, their performance was observed to be much lower compared to the their pre-trained/fine-tuned alternatives. Thus, there is a strong need for finding more efficient pre-training and fine-tuning schemes, which could lead to more cost-effective LLM-based VPMs.

IV. CONCLUSION

Transformer-based models and particularly LLMs, although originally proposed for natural text understanding and processing, are believed to revolutionize the way that various software engineering tasks are executed. Despite their recent proposition, a large body of research have already explored their potential utilization in critical software engineering tasks, including vulnerability prediction. The purpose of the present paper was to review the current research endeavors in utilizing LLMs for vulnerability prediction, in order to evaluate whether LLMs have solved the existing problems [48], and identify and report existing open issues and challenges that remain unresolved and require further research. Our work led to the observation that, although LLM-based VPMs have demonstrated remarkable results in vulnerability prediction, outperforming (and essentially replacing) more traditional and simpler DL-based approaches such as RNN and CNN, there are still many challenges that need to be addressed before being considered highly accurate, reliable, and practical.

ACKNOWLEDGMENT

Work reported in this paper has received funding from the EU's Horizon Europe Research and Innovation Program through the DOSS project, Grant Number 101120270.

REFERENCES

[1] M. Carvalho, J. DeMott, R. Ford, and D. A. Wheeler, "Heartbleed 101," *IEEE security & privacy*, vol. 12, no. 4, pp. 63–67, 2014.

[2] J. Luszcz, "Apache struts 2: how technical and development gaps caused the equifax breach," *Network Security*, vol. 2018, no. 1, pp. 5–8, 2018.

[3] Q. Chen and R. A. Bridges, "Automated behavioral analysis of malware: A case study of wannacry ransomware," in *2017 16th IEEE International Conference on machine learning and applications (ICMLA)*. IEEE, 2017, pp. 454–460.

[4] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[5] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.

[6] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *2014 IEEE 25th international symposium on software reliability engineering*. IEEE, 2014, pp. 23–33.

[7] H. K. Dam, T. Tran, T. T. M. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for predicting vulnerable software components," *IEEE Transactions on Software Engineering*, 2018.

[8] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *arXiv preprint arXiv:1909.03496*, 2019.

[9] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak, and L. Karaçay, "Vulnerability prediction from source code using machine learning," *IEEE Access*, vol. 8, pp. 150 672–150 684, 2020.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[11] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *arXiv:01367*, 2018.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI*, 2019.

[13] J. Devlin, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[14] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[15] S. Wang, L. Huang, A. Gao, J. Ge, T. Zhang, H. Feng, I. Satyarth, M. Li, H. Zhang, and V. Ng, "Machine/deep learning for software engineering: A systematic literature review," *IEEE Transactions on Software Engineering*, 2022.

[16] A. Bagheri and P. Hegedűs, "A comparison of different source code representation methods for vulnerability prediction in python," in *Quality of Information and Communications Technology*, 2021.

[17] X. Yuan, G. Lin, Y. Tai, and J. Zhang, "Deep neural embedding for software vulnerability discovery: Comparison and optimization," *Secur. Commun. Networks*, vol. 2022, pp. 5 203 217:1–5 203 217:12, 2022.

[18] K. Filus, P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe, "Efficient feature selection for static analysis vulnerability prediction," *Sensors*, vol. 21, no. 4, p. 1133, 2021.

[19] K. Filus, M. Siavvas, J. Domańska, and E. Gelenbe, "The random neural network as a bonding model for software vulnerability prediction," in *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems: 28th International Symposium, MASCOTS 2020*, 2021.

[20] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[21] I. Kalouptsoglou, M. Siavvas, D. Tsoukalas, and D. Kehagias, "Cross-project vulnerability prediction based on software metrics and deep learning," in *International Conf. on Comp. Sci. and its App.*, 2020.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv:1301.3781*, 2013.

[23] Y. Fang, Y. Liu, C. Huang, and L. Liu, "FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm," *Plos one*, 2020.

[24] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "VulDeePecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681*, 2018.

[25] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet?" *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3280–3296, 2022.

[26] I. Kalouptsoglou, M. Siavvas, A. Ampatzoglou, D. Kehagias, and A. Chatzigeorgiou, "Software vulnerability prediction: A systematic mapping study," *Information and Software Technology*, vol. 164, 2023.

[27] D. Coimbra, S. Reis, R. Abreu, C. Păsăreanu, and H. Erdogmus, "On using distributed representations of source code for the detection of c security vulnerabilities," *arXiv preprint arXiv:2106.01367*, 2021.

[28] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "Code2vec: Learning distributed representations of code," *Proc. ACM Program. Lang.*, 2019.

[29] H. Hanif and S. Maffeis, "VulBERTa: Simplified source code pre-training for vulnerability detection," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[31] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated vulnerability detection in source code using deep representation learning," in *17th IEEE international conference on machine learning and applications*, 2018.

[32] S. Kim, J. Choi, M. E. Ahmed, S. Nepal, and H. Kim, "VulDeBERT: A Vulnerability Detection System Using BERT," in *2022 IEEE International Symp. on Soft. Reliab. Eng. Workshops (ISSREW)*, 2022, pp. 69–74.

[33] N. Ziems and S. Wu, "Security vulnerability detection using deep learning natural language processing," in *2021-IEEE Conference on Computer Communications Workshops*. IEEE, 2021.

[34] B. Steenhoek, M. M. Rahman, R. Jiles, and W. Le, "An empirical study of deep learning models for vulnerability detection," *arXiv:08109*, 2022.

[35] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A C/C++ code vulnerability dataset with code changes and CVE summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 508–512.

[36] M. D. Purba, A. Ghosh, B. J. Radford, and B. Chu, "Software vulnerability detection using large language models," in *IEEE 34th International Symposium on Software Reliability Engineering Workshops*, 2023.

[37] G. Bhandari, A. Naseer, and L. Moonen, "CVEfixes: automated collection of vulnerabilities and their fixes from open-source software," in *17th International Conf. on Predictive Models and Data Analytics*, 2021.

[38] J. Zhang, C. Wang, A. Li, W. Sun, C. Zhang, W. Ma, and Y. Liu, "An empirical study of automated vulnerability localization with large language models," *arXiv preprint arXiv:2404.00287*, 2024.

[39] X. Zhou, T. Zhang, and D. Lo, "Large language model for vulnerability detection: Emerging results and future directions," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 2024, pp. 47–51.

[40] S. Pan, L. Bao, X. Xia, D. Lo, and S. Li, "Fine-grained commit-level vulnerability type prediction by cwe tree structure," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.

[41] M. Fu and C. Tantithamthavorn, "LineVul: A Transformer-based Line-Level Vulnerability Prediction," in *IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022.

[42] A. Sotgiu, M. Pintor, and B. Biggio, "Explainability-based debugging of machine learning for vulnerability discovery," in *17th International Conference on Availability, Reliability and Security*, 2022.

[43] A. Marchetto, "Can explainability and deep-learning be used for localizing vulnerabilities in source code?" in *5th ACM/IEEE International Conference on Automation of Software Test (AST 2024)*, 2024.

[44] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection," in *26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023.

[45] M. Fu, C. Tantithamthavorn, T. Le, Y. Kume, V. Nguyen, D. Phung, and J. Grundy, "AIBugHunter: A practical tool for predicting, classifying and repairing software vulnerabilities," *Empirical Software Engineering*, vol. 29, no. 1, p. 4, 2024.

[46] E. Gelenbe, M. Nakıp, and M. Siavvas, "System-wide vulnerability of multi-component software," *Computers & Industrial Engineering*, 2024.

[47] E. Gelenbe and M. Nakip, "IoT Network Cybersecurity Assessment With the Associated Random Neural Network," *IEEE Access*, vol. 11, pp. 85 501–85 512, 2023.

[48] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, "Static analysis-based approaches for secure software development," in *Security in Computer and Information Sciences*. Springer International Publishing, 2018, pp. 142–157.