Emerging Results in Using Explainable AI to Improve Software Vulnerability Prediction

Fahad Al Debeyan
f.aldebeyan@lancaster.ac.uk
School of Computing and Communications, Lancaster University
Lancaster, UK
College of Computer and Information Science, King Saud University
Riyadh, Saudi Arabia

Tracy Hall
tracy.hall@lancaster.ac.uk
School of Computing and Communications, Lancaster University
Lancaster, UK

Lech Madeyski
lech.madeyski@pwr.edu.pl
Faculty of Information and Communication Technology, Wroclaw University of Science and Technology
Wroclaw, Poland

ABSTRACT

Explainable Artificial Intelligence (XAI) has recently been applied to vulnerability prediction models to understand the decisions made and to improve the transparency of those models. We are the first to leverage XAI explanations to improve vulnerability prediction performance. The performance of vulnerability prediction models relies on the quality of the vulnerability dataset and the machine learning model. We use XAI information to identify biases in vulnerability prediction datasets and limitations in deep learning-based prediction models. Our XAI analysis is based on using a state-of-the-art deep-learning vulnerability prediction model (LineVul) and an explainability algorithm (Layered Integrated Gradients) to generate XAI information. The XAI information that we generated allowed us to improve our understanding of how our models worked, such that we were able to identify important improvement opportunities. Consequently, we present some surprising findings: while LineVul accurately predicted vulnerable functions, in 43% of cases, the use of XAI data allowed us to identify that those predictions were based on dataset biases rather than on actual vulnerable lines. By systematically removing these dataset biases, we achieved a notable performance improvement, increasing LineVul's F-Measure from 92% to 96%. Additionally, the insight we gained from XAI also allowed us to identify a fundamental limitation in LineVul's reliance on CodeBERT, a pre-trained language model limited to 512 tokens. By integrating LongCoder, a pre-trained model capable of processing longer sequences, we achieved an F-measure and MCC increase from 92% and 91%, respectively, to 94%, highlighting the potential for improved handling of complex, long-sequence vulnerabilities. We conclude that XAI has important additional applications that go beyond providing users with information describing the basis of predictions.

FSE Companion '25, June 23–28, 2025, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/25/06.
https://doi.org/10.1145/3696630.3728499

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Computing methodologies → Machine learning approaches**.

KEYWORDS

Explainable AI, Software Vulnerability Prediction

ACM Reference Format:
Fahad Al Debeyan, Tracy Hall, and Lech Madeyski. 2025. Emerging Results in Using Explainable AI to Improve Software Vulnerability Prediction. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728499

1 INTRODUCTION

With the availability of large databases and recent advances in deep learning techniques, AI systems are now achieving, and in some cases surpassing, human-level performance in a growing array of tasks. Nevertheless, deep learning-based models operate as black boxes, making it challenging to ascertain whether their effective performance stems from correct learning or if dataset biases are present that result in overly optimistic performance [3, 4].

Vulnerability prediction is one area where deep learning has helped make impressive strides in recent years. The latest state-of-the-art models, e.g. LineVul, have shown very good F1 scores of over 90% [10, 16]. However, our work uncovers startling evidence that these high scores mask underlying biases in a widely used dataset, leading to unreliable predictions. This raises critical questions about the validity of past research and underscores the urgent need for more rigorous approaches.

Explainable Artificial Intelligence (XAI) is *the process of providing explanations for the decisions made by an artificial intelligence model* [24]. XAI has been used to improve trust [19] and transparency [6, 27] in deep learning models in domains such as autonomous vehicles [26] and healthcare [1]. In contrast to prior work, we aim to leverage XAI not just for transparency but as a powerful tool to improve model performance and dataset quality, introducing a novel technique that bridges explanation and improvement. To achieve our aim, we answer the following research questions:

RQ1: Can explainability help in identifying potential bias in vulnerability prediction datasets? To ensure reliable predictions, the data to train and evaluate models must be unbiased. To answer RQ1, we analyse whether LineVul attributes correctly predicted vulnerabilities (true positives) to the code features causing the vulnerability or if predictions result from bias in the BigVul dataset [7]. We use the Layered Integrated Gradients algorithm [23] to extract the lines LineVul attributes to the vulnerable decision, compare these to the actual vulnerable lines, and measure the percentage agreement. If a model achieves a high F-Measure but a low line-level attribution score, bias in the dataset (e.g. duplicates or incorrect labelling) may be incorrectly helping the model differentiate between vulnerable and non-vulnerable instances.

```
1 void rs_filter_graph(RSFilter *filter) {
2     GString *str = g_string_new("digraph G
          ↪ {\n");
3     g_string_append_printf(str, "}\n");
4     g_file_set_contents("filter-graph", str->str, str->len,
          NULL);
5     if (0 != system("gnome-open/tmp/
          ↪ rs-filter-graph.png"))
6         g_warning("Calling gnome-openfailed.");
7 }
```

Listing 1: Output of a C++ function after applying the line-level explainer showing correct attributions (green), incorrect attributions (blue), or missed vulnerable line (red).

RQ2: Can explainability help in identifying limitations in deep learning vulnerability prediction models? It is important to have confidence that predictions are correct. However, the black-box nature of deep learning models makes identifying potential problems difficult. To answer RQ2, we focus on vulnerabilities missed by the model (false negatives). We examine explanations of vulnerable instances where LineVul failed to detect the vulnerability and look for common patterns in attributing features (lines) that led to incorrect classifications.

Since the original dataset, BigVul [7], keeps a record of the vulnerable lines, we can compare the results of the explanations to the vulnerable lines of code to understand whether or not the model is attributing the vulnerable lines to classification decisions. Listing 1 shows an example of a vulnerable function after applying the line-level explainer [15].

To our knowledge, this paper presents the first approach using XAI to help identify dataset bias and model limitations in software vulnerability prediction.

2 BACKGROUND

Tantithamthavorn and Jiarpakdee [25] introduced Explainable AI for Software Engineering (XAI4SE) to improve explainability in software engineering tasks. They published several papers on the explainability of defect prediction [12–14], a field closely related to vulnerability prediction, to help developers identify the characteristics of software defects and provide fixing suggestions.

LineVul [10] is a leading vulnerability prediction model that employs a transformer-based, line-level approach by integrating BERT's self-attention layers with CodeBERT to enhance vector representations with lexical and logical semantics. Utilising BERT's attention mechanism, LineVul identifies vulnerable lines, achieving a 91% F-measure for function-level prediction on the BigVul dataset, surpassing other models by 160–379%. For line-level prediction, it attained a 65% top-10 accuracy[1]. LineVul was selected for its superior performance, and enhancing it could significantly improve less effective models.

Layered Integrated Gradients (LIG) [23] provides a means of attributing importance scores to individual input features by approximating the integral of gradients of the model's output with respect to the inputs along a specified path. This enables a better understanding of which features have the most influence on the model's output. We have applied LIG to gain insights into the decision-making of deep learning vulnerability prediction models.

3 DESIGN AND EXPERIMENTAL SETUP

Given a vulnerability prediction dataset, instances are split into training and testing data. The training data consists of the source code of functions and the binary label for every function (vulnerable or non-vulnerable). The training data is used to train the vulnerability prediction model (LineVul). The test data comprises the source code of functions and the line-level ground truths (vulnerable lines) in every vulnerable function. The source code is used to evaluate the performance of the model. Then, using an explainability tool, the XAI algorithm produces an attribution score for every input token in the source code. We aggregate the scores for tokens in each line to produce line-level attributions. Then, we use the line-level ground truths to measure the correctness of the line-level attributions. Listing 1 shows a coloured version of vulnerable functions in the test data, where lines are coloured based on line-level attribution agreement with the ground truth.

To assess the quality of the vulnerability prediction dataset (RQ1), we examine instances where the model correctly predicts the vulnerability (true positive). As Chakraborty et al. [2] point out, a biased dataset can lead to overly optimistic model performance.

Dataset. LineVul is originally evaluated using the BigVul dataset from Fan et al. [7], which provides line-level ground truths indicating which lines in a function cause vulnerabilities. In contrast, other datasets like Devign [28] and Reveal [2] offer only function-level ground truths. BigVul is among the largest vulnerability datasets with line-level annotations, compiled from 348 open-source GitHub projects. It includes 91 CWE types from 2002 to 2019, encompassing 188,636 C/C++ functions with a 5.7% vulnerability rate (10,900 vulnerable functions) and 5,060,449 lines of code, of which 0.88% (44,603 lines) are vulnerable. Within the vulnerable functions, the percentage of vulnerable lines ranges from 2.5% to 20%, with a median of 7%. We use the BigVul dataset to analyse model explanations at the line level, employing the same data splits as LineVul's reproduction package: 80% training, 10% validation, and 10% testing. We run all models on the same test set to maintain evaluation consistency.

Line-Level Vulnerability Explanations. To answer our research questions, we rely on the vulnerability prediction model's explanations for each instance in the dataset. Such explanations

[1]Top-n Accuracy measures the percentage of vulnerable functions where at least one actual vulnerable line appears in the top n lines predicted vulnerable [18].

provide an understanding of the reason behind every prediction. Using the LIG algorithm [23], we perform attribution analysis on LineVul to extract the amount of contribution (a number between -1 and 1, where a positive number means a positive contribution) every input token (i.e., code word) has on the prediction decision. We chose LIG over other XAI algorithms like Saliency [21] and DeepLift [20] because the way LineVul is structured results in Saliency and DeepShift only producing positive attributions. Since we are interested in the line-level contributions, we follow existing techniques [10, 22] by aggregating the token contribution for every line in the function and considering the lines with a positive total as the lines contributing to the prediction. We publish our datasets and scripts in our reproduction package [9].

Table 1: LineVul reproduction on the original dataset.

Original Results			Reproduced Results		
Precision	Recall	F1	Precision	Recall	F1
0.97	0.86	0.91	0.96	0.88	0.92

4 RESULTS

To improve LineVul [10] and the BigVul dataset [7], we must ensure we reproduce the original results reported by LineVul. Table 1 shows the *precision*, *recall*, and *F-Measure* reported by LineVul compared to our reproduction results. Our reproduction results are within a 2% difference compared to the original paper in general.

In all subsequent models, we ran the evaluation 10 times to measure the standard deviation, the statistical significance using the Wilcoxon signed-rank test, and the effect size.

RQ1: Can explainability help in identifying potential bias in vulnerability prediction datasets? We focus on and collect the correctly predicted vulnerable functions in the test data. From 1055 vulnerable functions in the test data, LineVul correctly predicted 937 functions (89%). Then, for each of the 937 vulnerable functions, we compute the line-level attributions given by the explainability tool Captum and compare them to the line-level ground truths in the BigVul dataset. A correctly predicted vulnerable line is one that exists within the line-level attributions that influence the prediction positively.

We found that out of 937 correctly predicted vulnerable functions, LineVul attributed only 62% of the actual vulnerable lines to the prediction. This means that in 38% of cases, LineVul's predictions were not based on the actual vulnerable lines. Manual analysis revealed two potential biases. First, LineVul appeared to learn from lines containing vulnerability-prone assertions (e.g., *memory management*, *input validation*) that were not directly vulnerable. To address this, we incorporated patched versions of vulnerabilities, following Chakraborty et al. [2], so that the negative sample also includes vulnerability-prone assertions. While BigVul includes patched versions, they are in a field that LineVul does not use. We included them to improve the dataset's quality, though this only increased the negative sample by 5%. Additionally, in 39% of correctly predicted vulnerable functions, LineVul attributed at least one comment line to its decision. Since comment lines do not affect code behaviour, we removed all comments from the dataset to mitigate this bias. After removing comment lines, we removed vulnerable functions that showed no difference between the vulnerable code and the patched code (i.e. the original difference was in comment lines). Removing such vulnerable functions resulted in a reduction in vulnerable functions from 10,900 to 7,716.

Table 2 presents the results of LineVul on the original dataset as well as the dataset with biases removed (cleaned). The table shows the F-measure, MCC and line-level attribution. Table 2 shows that after removing bias from the BigVul dataset (cleaned), LineVul had an improved F-measure and MCC (from 92% and 91%, respectively, to 96%) with a standard deviation $\sigma < 0.1$ for both metrics. The Wilcoxon signed-rank test showed $W = 0$ and $p = 0.0019$ for F-measure and MCC, meaning the improved performance is statistically significant. Also, Cohen's d value showed $d > 4$ for both F-measure and MCC, suggesting a strong effect. Additionally, the line-level attribution rises from 62% to 68%. This suggests that the cleaned BigVul dataset not only improved the performance of LineVul but also increased the line-level attribution percentage, which shows that the model should be more reliable in practice. The findings indicate that explainability can aid in identifying potential bias in vulnerability prediction datasets.

Table 2: Performance measures of LineVul evaluated on the original BigVul dataset and the cleaned version.

BigVul Version	F1	MCC	Line-level Attr.
Original	0.92	0.91	62%
Cleaned	0.96	0.96	68%

RQ2: Can explainability help in identifying limitations in deep learning vulnerability prediction models? To answer RQ2, we focus on the false negatives in the test data. First, we collect the vulnerable functions that were incorrectly predicted. LineVul predicted 118 (11%) of the 1055 vulnerable functions in the test data incorrectly (false negatives). Then, for each of the 118 vulnerable functions, we compute the line-level attributions generated by the explainability tool Captum. Because LineVul inaccurately predicted the functions as non-vulnerable, the positive line-level attributions give us the lines that attribute to the non-vulnerable prediction. LineVul accurately predicted 17,771 (99.8%) out of 17,809 non-vulnerable functions. With such a minimal false positive rate of 0.2%, we decided not to focus on strategies for reducing false positives.

We manually examined the explanations for the 118 incorrectly predicted vulnerable functions and discovered that the primary cause of the incorrect prediction was that LineVul uses a CodeBERT [8] pre-trained language model to generate vector representations of source code, which only preserves the first 512 tokens from each function. We found that *69% of vulnerable lines in the BigVul dataset occur outside the 512 token limit*. Therefore, instead of CodeBERT, we decided to use the LongCoder [11] pre-trained language model, which can generate a vector representation of source code up to 4096 tokens to improve LineVul's performance. Due to memory resource constraints, we limited the maximum token vector size to 1024. Increasing the vector size above 1024 may potentially result in further improvement in model performance.

Table 3 presents the results of evaluating LineVul and LineVul+LongCoder on the BigVul dataset. While keeping precision at 96%, using LongCoder to generate a vector representation of source code increased the recall from 88% to 93%, resulting in an increased F-measure and MCC from 92% and 91%, respectively, to 94% with a standard deviation $\sigma < 0.1$ for both metrics. The Wilcoxon signed-rank test showed $W = 0$ and $p = 0.0019$, meaning the improved performance is statistically significant. Also, Cohen's d value showed $d > 3.8$ for both F-measure and MCC, suggesting a strong effect. The results show that using LongCoder improves the performance of LineVul.

Table 3: Performance measures of LineVul and LinveVul + LongCoder evaluated on the original BigVul dataset.

Model	Precision	Recall	F1	MCC
CodeBERT (original)	0.96	0.88	0.92	0.91
LongCoder	0.96	0.93	0.94	0.94

Table 4: Performance measures of LineVul using LongCoder evaluated on BigVul (cleaned).

Model	Precision	Recall	F1	MCC
CodeBERT (original)	0.98	0.95	0.96	0.96
LongCoder	0.99	0.94	0.96	0.96

Table 4 presents the results of evaluating LineVul and LineVul+LongCoder on the cleaned BigVul. While precision slightly increased from 98% to 99% using LongCoder, the recall decreased from 95% to 94%, resulting in a similar F-measure and MCC. These results indicate that incorporating LongCoder did not affect the overall performance of LineVul trained on the cleaned BigVul dataset.

The literature discusses the inclusion of patched versions of vulnerable functions in the training set and CodeBERT's token size limitation [2, 5]. However, the effects of these limitations on LineVul could only be quantified using Explainable AI. Our XAI analysis showed that for 39% of correctly predicted vulnerable functions, LineVul based its decisions on comment lines. Additionally, XAI allowed us to measure that 68% of vulnerable lines were beyond the 512 token limit set by CodeBERT.

5 RELATED WORK

Recent studies in software vulnerability prediction [10, 22] have applied explainability techniques for various purposes. LineVul [10] uses explainability to help developers identify potential causes of vulnerable functions. Steenhoek et al. [22] surveyed nine deep learning models on popular datasets (Devign and BigVul) and analysed model explanations and important features used for predictions. Although this paper also focuses on model explanations, we use them to improve the performance of vulnerability prediction models.

Chakraborty et al. [2] evaluated four vulnerability prediction models—ReVeal [2], VulDeePecker [17], SySeVR [16], and Devign [28] and found that their training datasets often contained synthetic or duplicate data. These training sets led the models to learn irrelevant artefacts, such as variable or function names, instead of actual vulnerability causes. When re-evaluated using real-world Github data, the models' performance declined significantly. Our work applies XAI to further expose potential biases in vulnerability prediction datasets.

Al Debeyan et al. [5] investigated how different types of negative instances in vulnerability prediction datasets affect model performance. They distinguished between easy negatives (largely different from positive samples) and hard negatives (sharing many characteristics with positives). Their findings show that datasets containing only one type of negative instance can introduce bias, leading to overly optimistic results. They also found that a 1:15 ratio of hard to easy negatives yields the best performance for predicting vulnerabilities across projects. Our work introduces a new method for identifying bias in such datasets.

6 LIMITATIONS

In this paper, we employ the Integrated Gradient algorithm to generate line-level attributions for model predictions. It is important to note that these attributions may vary depending on the specific algorithm and the explainability tool utilised, potentially leading to different conclusions. We tried using other explainability algorithms like Saliency maps and DeepLift, but due to the architecture of LineVul, these algorithms were only producing positive attributions, limiting the ability for negative attribution analysis. Consequently, suggested improvements may not consistently improve either the dataset or the model under test and thus must be evaluated thoroughly.

On the other hand, using XAI can be beneficial in identifying widely occurring patterns that result in dataset bias or model limitations. However, deriving limitations from a limited set of instances could potentially lead to model overfitting.

Finally, in this paper, we utilised changed lines from vulnerability fixes as vulnerable lines. However, changed lines may not always be the root cause of the vulnerability. We acknowledge that a dataset encompassing root cause lines could potentially impede the achievement of more favourable results; however, to the best of our knowledge, there are currently no available vulnerability datasets that incorporate root cause lines.

7 FUTURE PLANS

Our work focused on using XAI to help identify bias in the BigVul dataset and to identify limitations in LineVul. We focused on true positives and false negatives to identify potential bias in datasets and limitations in deep learning models. We plan to extend our work to examine the other two quadrants (false positives and true negatives) as they may reveal other limitations in deep learning models. Additionally, we plan to extend our work by examining a wider list of datasets and vulnerability prediction models and investigating whether further improvements can be achieved on lower-performing vulnerability prediction models. We also plan to extend the work to eliminate the need for manual inspection of explanations by using AI to automatically find common patterns that limit the performance of deep learning models.

We also intend to expand our research to provide a framework for using explainability to improve the performance of machine learning models for other software engineering tasks, such as defect prediction and automatic program repair. We envision building a scalable system that, when given a dataset and a model as inputs, offers insights into potential dataset biases and model limitations using explainability.

8 CONCLUSION

In this paper, we explore using explainability to enhance the performance of deep learning vulnerability prediction models. By using the Layered Integrated Gradients algorithm to calculate line-level attributions for each prediction, we were able to identify potential bias in the dataset as well as identify limitations to a deep learning vulnerability prediction model. Taking the LineVul model and the BigVul dataset as a use case, we found that although LineVul accurately predicted vulnerable functions, LineVul only attributed the vulnerable prediction to actual vulnerable lines in 57% of cases. We identified two biases in how LineVul uses BigVul: 1) LineVul does not include patched versions of vulnerable functions, and 2) LineVul attributed at least one comment line to 39% of its vulnerable predictions. Removing biases resulted in improving the F-measure of LineVul from 92% to 96%. Our study has also revealed a limitation in LineVul. We found that LineVul's use of a limited vector size from the CodeBERT language model led to incorrect predictions. By incorporating a more comprehensive language model (LongCoder), we increased the F-measure and MCC of LineVul from 92% and 91%, respectively, to 94% on the original BigVul dataset. These findings show a greater benefit from removing dataset bias compared to increasing the vector size limit for LongCoder. It is worth exploring these effects on other lower-performing vulnerability prediction models. These findings not only validate the transformative potential of XAI but also signal a paradigm shift in software vulnerability prediction research.

ACKNOWLEDGMENT

Lech Madeyski initiated the work on this topic during his research internship at Lancaster University at the invitation of Prof. Tracy Hall. Lech Madeyski was partially financially supported by an Engineering and Physical Sciences Research Council (EPSRC) grant EP/S005730/1.

REFERENCES

[1] Julia Amann, Alessandro Blasimme, Effy Vayena, Dietmar Frey, and Vince I Madai. 2020. Explainability for artificial intelligence in healthcare: a multidisciplinary perspective. *BMC medical informatics and decision making* 20, 1 (2020), 1–9.

[2] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2022. Deep Learning Based Vulnerability Detection: Are We There Yet? *IEEE Transactions on Software Engineering* 48, 9 (2022), 3280–3296. https://doi.org/10.1109/TSE.2021.3087402

[3] Robert Challen, Joshua Denny, Martin Pitt, Luke Gompels, Tom Edwards, and Krasimira Tsaneva-Atanasova. 2019. Artificial intelligence, bias and clinical safety. *BMJ Quality & Safety* 28, 3 (2019), 231–237.

[4] Lieyang Chen, Anthony Cruz, Steven Ramsey, Callum J Dickson, Jose S Duca, Viktor Hornak, David R Koes, and Tom Kurtzman. 2019. Hidden bias in the DUD-E dataset leads to misleading performance of deep learning in structure-based virtual screening. *PloS one* 14, 8 (2019), e0220113.

[5] Fahad Debeyan, Lech Madeyski, Tracy Hall, and David Bowes. 2024. The Impact of hard and easy negative training data on vulnerability prediction performance. *Journal of Systems and Software* (2024), 112003.

[6] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2009. Visualizing higher-layer features of a deep network. *University of Montreal* 1341, 3 (2009), 1.

[7] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 508–512. https://doi.org/10.1145/3379597.3387501

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[9] Anonymous for review. 2024. Replication Package for the Paper Titled "Emerging Results in Using Explainable AI to Improve Software Vulnerability Prediction". https://doi.org/10.5281/zenodo.13902312

[10] Michael Fu and Chakkrit Tantithamthavorn. 2022. LineVul: A Transformer-Based Line-Level Vulnerability Prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 608–620.

[11] Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. LongCoder: A Long-Range Pre-trained Language Model for Code Completion. *arXiv preprint arXiv:2306.14893* (2023).

[12] J Jiarpakdee, C Tantithamthavorn, and AE Hassan. 2019. The impact of correlated metrics on the interpretation of defect prediction models. *IEEE Trans Softw Eng Early Access* (2019).

[13] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. 2018. AutoSpearman: Automatically mitigating correlated software metrics for interpreting defect models. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, 92–103.

[14] Jirayus Jiarpakdee, Chakkrit Kla Tantithamthavorn, Hoa Khanh Dam, and John Grundy. 2020. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering* 48, 1 (2020), 166–185.

[15] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for PyTorch. *CoRR* abs/2009.07896 (2020). arXiv:2009.07896 https://arxiv.org/abs/2009.07896

[16] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. 2018. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. *CoRR* abs/1807.06756 (2018).

[17] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. *CoRR* abs/1801.01681 (2018).

[18] Felix Petersen, Hilde Kuehne, Christian Borgelt, and Oliver Deussen. 2022. Differentiable Top-k Classification Learning. arXiv:2206.07290 [cs.LG]

[19] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[20] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International conference on machine learning*. PMLR, 3145–3153.

[21] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034* (2013).

[22] Benjamin Steenhoek, Md Mahbubur Rahman, Richard Jiles, and Wei Le. 2023. An empirical study of deep learning models for vulnerability detection. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2237–2248.

[23] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*. PMLR, 3319–3328.

[24] Chakkrit Tantithamthavorn and Jirayus Jiarpakdee. 2021. *Explainable AI for Software Engineering*. Monash University. https://doi.org/10.5281/zenodo.4769127 Retrieved 2021-05-17.

[25] Chakkrit Kla Tantithamthavorn and Jirayus Jiarpakdee. 2021. Explainable ai for software engineering. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1–2.

[26] Éloi Zablocki, Hédi Ben-Younes, Patrick Pérez, and Matthieu Cord. 2022. Explainability of deep vision-based autonomous driving systems: Review and challenges. *International Journal of Computer Vision* 130, 10 (2022), 2425–2452.

[27] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, 818–833.

[28] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems* 32 (2019).