Cross-Domain Vulnerability Detection using LLMs: Knowledge Transfer from Contemporary Software to AI/ML-Specific Software

Miltiadis Siavvas*†, Ilias Kalouptsoglou*, Dionysios Kehagias*, and Dimitrios Tzovaras*
* Centre for Research and Technology Hellas/Information Technologies Institute, Thessaloniki, Greece
† Aristotle University of Thessaloniki, Department of Electrical and Electronic Engineering, Greece
siavvasm@iti.gr, iliaskaloup@iti.gr, diok@iti.gr, dimitrios.tzovaras@iti.gr

***Abstract*—The increasing integration of artificial intelligence (AI), and particularly machine learning (ML), into software-intensive systems raises security concerns for AI/ML-specific code. Although AI-based vulnerability detection models (VDMs) achieve accurate prediction on contemporary software, their cross-domain effectiveness on AI/ML-specific software remains underexplored due to limited labeled data. To fill this gap, in this paper we aim to examine whether VDMs trained on contemporary software can accurately detect vulnerabilities in AI/ML-specific software as well, and to assess how domain adaptation can affect their performance. We construct a dataset of vulnerable and non-vulnerable Python functions split into contemporary and AI/ML-specific subsets, fine-tune CodeBERT and CodeGPT on the contemporary subset, and evaluate them on the AI/ML subset. We then perform gradual adaptation by adding 5%, 10%, 15%, 20%, 25%, 40%, and 50% target-domain samples. We also analyze representation drift with UMAP and attention drift via self-attention-based explainability. Results suggest that AI-based VDMs built on contemporary software cannot be used effectively on AI/ML-specific software without proper domain adaptation, indicating that AI/ML-specific software exhibits unique characteristics; however, even a small-moderate amount of domain-specific data can enable satisfactory cross-domain detection accuracy.**

***Index Terms*—Software Security, Vulnerability Detection, Large Language Models, Artificial Intelligence, Domain Adaptation, Explainable AI (XAI)**

I. Introduction

Software security remains critical since even a single vulnerability can have serious consequences for users, organizations, and critical infrastructures (e.g., [1]). At the same time, the rapid adoption of Artificial Intelligence (AI) and Machine Learning (ML), accelerated by the Transformer architecture [2] and Large Language Models (LLMs), has increased the amount of AI/ML-specific software deployed in important environments. As contemporary systems increasingly delegate key functionality to AI/ML models [3], the early detection of vulnerabilities in AI/ML software becomes essential [4].

Various software vulnerability detection techniques have been proposed over the years [5]. Among them, AI-based Vulnerability Detection Models (VDMs) have gained a particular attention since they can learn complex patterns directly from source code or its representations [6], [7]. Recently, pretrained Transformer-based models such as CodeBERT [8] have shown strong within-domain performance [6], [7], [9].

However, cross-project and cross-domain generalization remains a known weakness of AI-based VDMs [5]–[7], [9], [10]. This issue is largely unexplored for AI/ML-specific software. Specifically, it is unclear whether VDMs trained on contemporary software transfer to AI/ML code, how much domain adaptation is needed, and whether AI/ML-specific software introduces a substantial domain shift. This matters because AI/ML systems often rely on Python and specialized libraries such as TensorFlow and PyTorch, which introduce APIs and code patterns that may influence vulnerability proneness [4].

The main objective of the present paper is to examine the effectiveness of AI-based vulnerability detection models trained on contemporary software when applied to AI/ML-specific software, assess the role of domain adaptation in improving the detection accuracy, and investigate the amount of data required to render the performance of the model acceptable. In particular, this work investigates whether AI/ML-specific software exhibits unique characteristics that influence their vulnerability proneness, and whether these characteristics need to be considered during vulnerability detection. The goal of the paper can be expressed in the following Research Question:

RQ: Can AI-based vulnerability detection models trained on vulnerability data retrieved from contemporary software accurately detect vulnerabilities in AI/ML-specific software, and how does domain adaptation affect their performance?

To this end, in the present paper we conduct an empirical study to examine whether vulnerability-related knowledge learned from contemporary software can be transferred to the AI/ML-specific software, and what factors influence this transferability. To achieve this, we construct a dataset of vulnerable and non-vulnerable Python functions split into contemporary and AI/ML-specific subsets. We then fine-tune CodeBERT and CodeGPT [8] on the contemporary subset, evaluate it on the AI/ML subset, and apply gradual domain adaptation by incrementally adding target-domain samples. We complement the performance analysis with UMAP-based representation analysis and self-attention-based explainability to study representation and attention drift.

II. Related Work

In an attempt to detect more complex vulnerabilities than conventional static analysis techniques, a wide range of AI-

based VDMs have been proposed over the years, ranging from simple ML techniques based on software metrics to highly complex text mining-based deep learning (DL) models [5], [7]. Recently, the attention of the research community has shifted towards the adoption of Large Language Models (LLMs), which have demonstrated promising results in vulnerability detection (via proper adaptation) generally outperforming state-of-the-art models [11]. However, despite their promising results, LLM-based VDMs exhibit a set of weaknesses that need to be addressed [9]. One important weakness is their poor generalizability, and particularly their poor predictive performance in a cross-project and cross-domain setting. Particularly, while they are able to accurately detect vulnerabilities in code repositories they have been exposed to during their training, their predictive performance degrades significantly when employed to previously unknown projects or projects stemming from different domains.

As a solution to this issue, domain adaptation techniques have been examined. in which the original model is fed with data from the target domain, either labeled or unlabeled, in order to enable it to learn the target domain [7]. Prior work focused on reducing distributional discrepancies by aligning source and target code representations in a shared feature space, boosting-style reweighting with limited target labels, or via using unlabeled target data (i.e., code snippets) with richer code representations (e.g., code property graphs and GNNs) [12]–[14]. However, existing attempts focus on generic domain adaptation. To the best of our knowledge, no attempt exists focusing particularly on the AI/ML domain. This is important, since recent studies have shown that AI/ML-specific libraries and frameworks may contain important vulnerabilities [4].

Our work attempts to fill this gap by examining how AI-based VDMs built on contemporary software perform when employed to AI/ML-specific software, and how domain adaptation could affect their cross-domain performance. We also examine representation and attention drift by employing visualization and explainability techniques. We should clarify that we did not compare different domain adaptation techniques since our goal is not to propose the best adaptation method, but to examine whether VDMs trained on conventional software transfer to AI/ML-specific software and whether limited labeled target-domain data are sufficient for adaptation.

III. Methodology

The methodology of the present work is shown in Figure 1. A detailed description is provided in the rest of the section.

A. Dataset Construction

The first step is the construction of a suitable dataset. We used an extension [15] of the dataset proposed by Bagheri et al. [16], which contains a large number of vulnerable and clean Python files and functions. The dataset was constructed by crawling Python projects found on GitHub searching for vulnerability-fixing commits (i.e., commits that contain keywords indicating the patch of a specific vulnerability), with the version of the source code prior to the vulnerability fix being labeled as vulnerable, and the latest version of the same source code (after the fix) being labeled as non-vulnerable. We decided to focus on Python, since it is the most widely used language for building AI/ML models and applications.

This dataset was then divided into two individual subsets, i.e., one containing contemporary software functions, and another containing AI/ML-specific functions. To achieve this, we created a script that receives the source code of each Python function and searches for indicators of AI/ML code, such as the utilization of well-established methods (e.g., fit, predict, train, eval, DataLoader, optimizer, tokenizer, etc.) of popular AI/ML frameworks and libraries (e.g., TensorFlow, PyTorch, etc.). The script generates a score, indicating the percentage of AI/ML related code in the given function. We decided to add a given function to the AI/ML group when the percentage of AI/ML code exceeded a specific threshold, which was determined heuristically based on the authors' domain expertise. To validate this choice, we manually inspected a representative sample of 200 functions classified as AI/ML-specific and did not observe any false-positive classifications.

This process led to a dataset comprising: (i) 50,825 generic Python functions (14,918 vulnerable and 35,907 benign), and (ii) 1,622 AI/ML-specific functions (239 vulnerable and 1,383 benign). For the experimentation, the dataset was split into ML and non-ML samples, with 50% of ML data reserved as a test set. A configurable stratified subset of the remaining ML data was injected into training and combined with all non-ML samples to form the train/validation pool, which was then split 90/10 in a stratified manner.

B. Model Construction and Evaluation

For function-level vulnerability detection, we use the pre-trained Transformer-based models CodeBERT and CodeGPT-2 [8], [17], which follow encoder-only and decoder-only architectures respectively, and have demonstrated promising results in the literature [18]–[21] thus allowing us to study cross-domain adaptation across two representative code LLM families. In particular, as shown in Figure 1, we fine-tune both models for binary classification of contemporary software functions to vulnerable or clean, where the functions are tokenized with the corresponding tokenizer and passed to the model, while a classification head produces final prediction.

To evaluate the produced VDMs in both within-domain and cross-domain setting, we use common performance metrics, namely Recall, Precision, and F_1-Score, focusing mainly on F_1-Score since it balances Recall and Precision and is the most widely used metric in the related literature [5]. Due to the high imbalance of the dataset, Accuracy is omitted, since it will be unreliable potentially confusing the reader. The evaluation includes two settings: (i) the *within-domain setting*, where the models are trained and tested on the contemporary-software subset, and (ii) the *cross-domain setting*, where they are trained on contemporary software and tested on the AI/ML-specific subset, enabling comparison against an unseen domain.

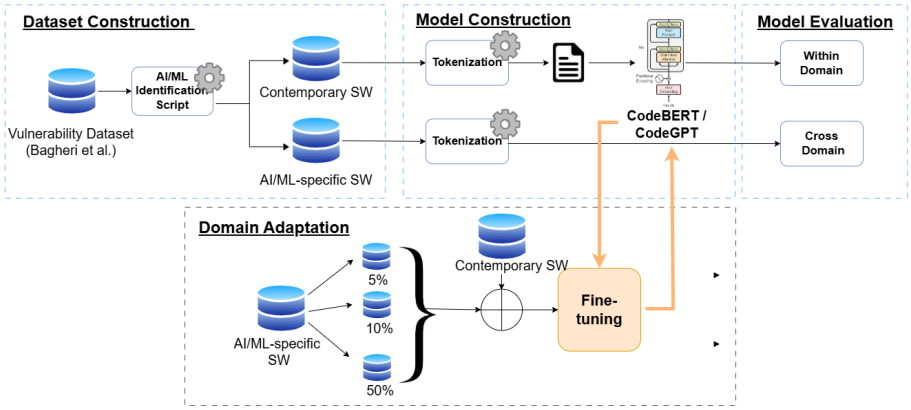


Fig. 1. Methodology of gradual domain adaptation.

C. Domain Adaptation Scheme

1) Gradual Domain Adaptation: As shown in Figure 1, after evaluating the predictive performance of the AI-based VDM in within-domain and cross-domain setting, the next step is to employ domain adaptation, in order to examine how incorporating domain-specific data from the target domain (i.e., the AI/ML-specific software subset) affects cross-domain detection accuracy. To obtain a clearer and more fine-grained view of this effect, we adopt a gradual adaptation scheme, in which we incrementally augment the set used for fine-tuning the Transformer-based VDM with progressively larger portions of the AI/ML-specific subset, and we re-fine-tune and re-evaluate the model at each step. In particular, we consider augmentation levels of 5%, 10%, 15%, 20%, 25%, 40%, and 50% of the target-domain samples. This enables us to empirically identify (i) the point at which cross-domain performance becomes satisfactory (i.e., the minimum amount of target-domain data required to achieve decent detection accuracy), and (ii) the point beyond which additional target-domain samples yield diminishing returns (i.e., only marginal improvements in the performance metrics).

2) Representation Drift: To gain a deeper understanding of how the feature space evolves during gradual domain adaptation, we conduct a representation analysis, to explore potential representation drift. More specifically, we analyze representation shift using penultimate-layer hidden states, which capture task-specific semantic information. We extract the embeddings of the penultimate layer (i.e., the last hidden Transformer layer right before the classification head) via CLS Token Pooling. CLS is a special classification token whose final embedding is used as the aggregate representation for prediction. Then we visualize them using Uniform Manifold Approximation and Projection (UMAP) [22]. UMAP is a technique that enables the projection of high-dimensional representations into a low-dimensional space (typically two dimensions) while aiming to preserve the local neighborhood structure of the original feature space, thus, facilitating the visual inspection of clustering and separability patterns. So, in our case, the UMAP plot of the CodeBERT and CodeGPT embeddings at each sequential and incremental adaptation step will enable the observation of how the learned representations drift as target-domain samples are gradually incorporated, and whether the vulnerable and non-vulnerable AI/ML-specific functions become progressively more separable (i.e., form more distinct clusters), reflecting improved alignment of the model's feature space with the target domain.

3) Attention Drift: We also use explainability (XAI) analysis, based on the self-attention mechanism [2] of the CodeBERT model, in order to examine how the feature importance of the model changes after each adaptation step, and whether unique salient tokens exist that influence the predictive performance of the model in the cross-domain setting. In particular, by leveraging the self-attention scores of CodeBERT and CodeGPT, we identify the top-20 most important features (i.e., tokens) for each function of the test set. We then compute how frequently each token appears across all functions of the test set (i.e., across all per-function top-20 lists) and derive the 20 most frequent salient tokens for the corresponding model checkpoint. We repeat this same process for each augmentation step, leading to a sequence of individual lists of dominant tokens. Then, we quantitatively compute the percentage of unique tokens that are added (i.e., newly introduced) in each augmentation step compared to the original one. This quantitative analysis allows us to observe how the salient tokens evolve after each augmentation (i.e., adaptation) step and how this evolution relates to the changes in the performance metrics.

IV. Results and Discussion

The results of our study are collectively presented in Table I. The last row of the table (i.e., the within-domain row) corresponds to the performance metrics of the model that was trained and tested solely on contemporary software, whereas the first row of the table (i.e., the cross-domain row) corresponds to the results of the model that was trained on contemporary software and tested on AI/ML-specific software. Intermediate rows report performance when progressively adding 5–50% target-domain samples during model fine-tuning.

As can be seen in Table I, the predictive performance both of the CodeBERT and the CodeGPT models in within-domain

TABLE I
PERFORMANCE OF THE CODEBERT AND CODEGPT MODELS UNDER GRADUAL DOMAIN ADAPTATION SETTINGS.

	CodeBERT			CodeGPT		
Setting	**F_1**	**Recall**	**Precision**	**F_1**	**Recall**	**Precision**
No Adapt.	27.45%	23.43%	33.14%	30.64%	22.18%	49.53%
5% Adapt.	23.16%	18.33%	31.43%	53.81%	44.17%	68.83%
10% Adapt.	59.33%	51.67%	69.66%	71.79%	70.00%	73.68%
15% Adapt.	68.83%	70.83%	66.93%	71.23%	65.00%	78.79%
20% Adapt.	72.50%	72.50%	72.50%	72.48%	65.83%	80.61%
25% Adapt.	72.81%	65.83%	81.44%	76.86%	73.33%	80.73%
40% Adapt.	76.52%	73.33%	80.00%	82.35%	81.67%	83.05%
50% Adapt.	82.10%	78.33%	86.24%	83.40%	81.67%	85.22%
Within	92.36%	91.55%	93.18%	91.76%	91.49%	92.04%

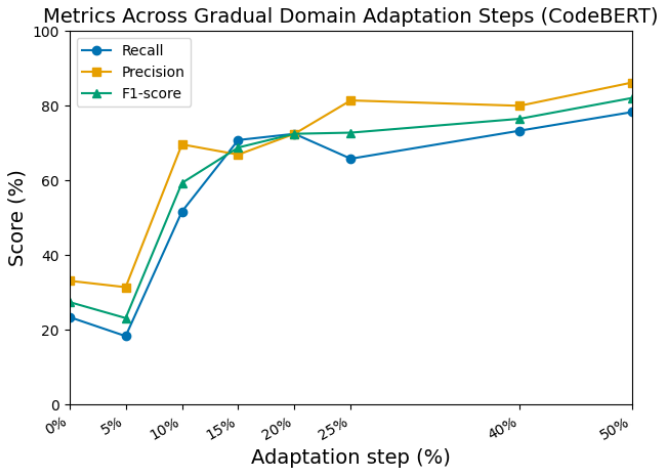


Fig. 2. Metrics across gradual domain adaptation steps.

setting (i.e., when tested and evaluated solely on contemporary software) was found to be really high, with all metrics being above 90%, and particularly F_1-Score being 92.36% and 91.76% respectively. When the models were tested in the cross-domain setting, i.e., when employed directly on the samples of the AI/ML-specific subset, the detection accuracy dropped significantly, with the F_1-Score being 27.45% for CodeBERT and 30.64% for CodeGPT. A similar behavior is observed with the other performance metrics, with Recall dropping to 23.43% for CodeBERT and 22.18% for CodeGPT, and Precision dropping to 33.14% and 49.53% respectively.

As shown in Table I, the incremental incorporation of target-domain data led to a steady improvement in the performance metrics of the models. For instance, the F_1-Score followed an upward trend and steadily increased across the considered adaptation steps, showing an incremental improvement. The only exception was the first adaptation step for CodeBERT, in which F_1-Score dropped from 27.45% to 23.16%, and the third adaptation step in CodeGPT in which the score remained almost the same. In addition to that, it seems that at around 10% augmentation for CodeGPT and 20% augmentation for CodeBERT the models' performance increases to acceptable level, as the F_1-Score becomes higher than 70% in both cases, whereas from that point and beyond the effectiveness of the augmentation becomes less pronounced, as the increase in performance metrics becomes smaller for the same (or even higher) amount of augmentation. The same behavior is observed for the other metrics as well.

To better understand this trend, the performance metrics across the considered adaptation steps for the CodeBERT model are plotted on the same line chart in Figure 2. It is clear that the incremental incorporation of target-domain samples lead to an increase in the performance of the model, with the increase being higher during the initial steps (e.g., up to the 15-20% step), and less pronounced in the later steps (the curves seem to plateau).

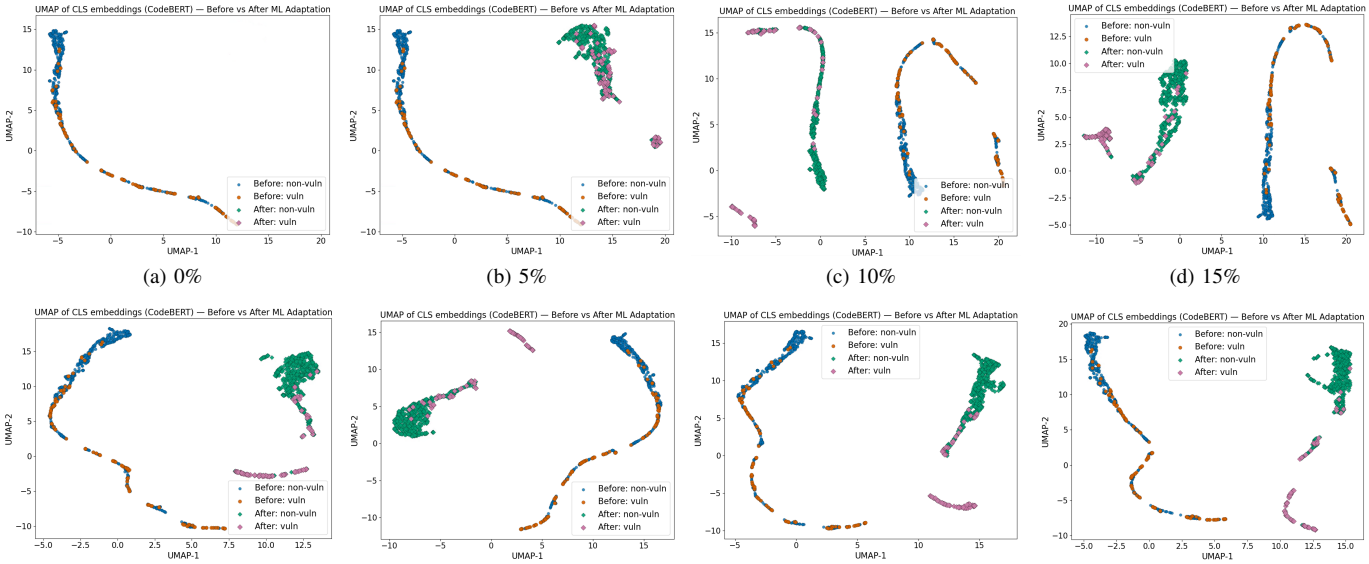
The slight degradation in CodeBERT performance at the 5% adaptation step can be attributed to the very small number of vulnerable target-domain data (approximately 12), which may have led to a temporary disruption of its decision boundary and temporarily induce negative transfer. However, it should be noted that this appears to be a model-specific instability rather than a general effect of domain adaptation, since, in contrast, CodeGPT benefits already from this small amount of target domain, which suggests that this behavior is architecture- and pretraining-dependent. Importantly, as more target-domain samples are incorporated, both models follow the expected upward performance trend, indicating that the initial fluctuation is transient and does not alter the main observation.

It should be noted that the values reported in Table I represent averages over five trials with different random seeds. We also performed the Wilcoxon signed-rank test between each adaptation step and, in all cases, obtained p-values below the 0.05 threshold, indicating statistically significant differences at the 95% confidence level.

In Figure 3, the results of the representation drift analysis are presented. As described in Section III-C2, each sub-figure visualizes the 2D UMAP projection of embedding representations extracted from the last hidden layer (i.e., penultimate layer) of the CodeBERT-based VDM after fine-tuning with the corresponding portion of AI/ML-specific samples. UMAP is employed to provide an intuitive visualization of the high-dimensional feature space, enabling the inspection of class separability patterns. It should be noted that for reasons of brevity, only the plots of CodeBERT model are provided. The UMAP plots of CodeGPT, which actually demonstrate a similar behavior, are available on the website[1] with the supporting material of the present work.

Each point in Figure 3 corresponds to an AI/ML test function and is colored by its ground-truth label (i.e., vulnerable vs. benign). As can be seen from the plots shown in Figure 3 (a), initially, the feature space was inseparable, justifying the very low F_1-Score that was observed previously (27.45%). Next, in Figure 3 (b)-(h) it is clear that as the proportion of target-domain data increases, the projected representations exhibit a clear drift and progressively improved class separability,

[1]https://sites.google.com/view/vulnerabilitydetectiononmldoma/



(a) 0% (b) 5% (c) 10% (d) 15%

(e) 20% (f) 25% (g) 40% (h) 50%

Fig. 3. UMAP-based representation drift analysis under gradual domain adaptation.

indicating that incorporating AI/ML-specific samples helps the model learn more discriminative, domain-relevant features.

This trend aligns with the corresponding improvements in model performance presented in Table I, suggesting that domain adaptation facilitates the learning of more discriminative, domain-relevant features. While the analysis is primarily qualitative, the observed evolution of the embedding space indicates an improved separation between class distributions.

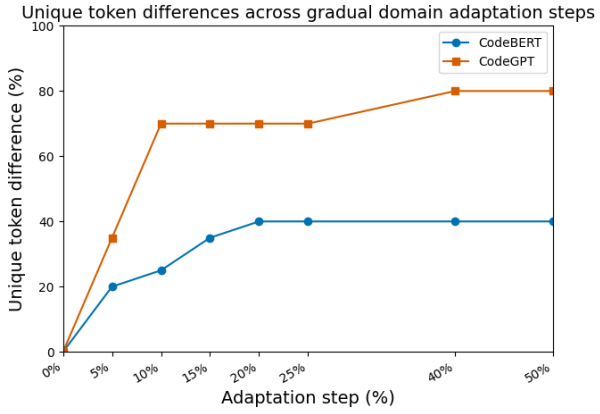


Fig. 4. Attention drift across adaptation steps

As stated in Section III-C3, the representation drift analysis is complemented by a feature (i.e., token) attention drift analysis. Following the procedure described in Section III-C3, we identified the 20 most frequent tokens of the top-20 features of the CodeBERT-based and CodeGPT-based VDMs for the AI/ML-specific subset for each adaptation step, and for all the adaptation steps the percentage difference of unique tokens compared to the original VDMs trained solely on contemporary software were computed, and are shown in Figure 4.

As shown in Figure 4, the incremental addition of target-domain data leads to a considerable change in the list of the top-20 most important features of the studied models. Specifically, for CodeBERT from 20% adaptation and beyond, the percentage difference is fixed to 40%, whereas for the CodeGPT model from 10% adaptation and beyond the percentage difference is above 75%, maxing at 80% at 40% adaptation. This suggests that there is an obsevable drift in feature importance (i.e., in the attention of the models to specific features/tokens), and that even a small (around 10% for CodeGPT) to moderate (around 20% for CodeBERT) augmentation, led to a substantial difference in the top-20 most important features of the model (40% for CodeBERT and 75% for CodeGPT). Notably, these breaking points in the salient tokens coincide with the points at which the models' performance surpasses the 70% F_1-Score, which is considered decent for cross-domain setting (as shown in Table I). Another interesting observation is that CodeGPT demonstrates larger percentage differences in each adaptation step compared to CodeBERT, which is inline with its observed faster and more abrupt domain adaptation, as shown in Table I.

This result also indicates that there may be domain-specific characteristics (e.g., library calls, tensor operations, and model-related functions such as fit, predict, and optimizer) that are highly important for the performance of the model in the given domain. This strengthens the belief that AI/ML-specific subset contain unique features (tokens) that may affect the detection accuracy of the model and need to be taken into account during model adaptation. At the same time, generic programming tokens remain important, but their relative importance appears to change, suggesting a different contextual role within AI/ML software.

Overall, the results demonstrate that AI/ML-specific soft-

ware seems to exhibit unique characteristics that are different from contemporary software with respect to the existence of vulnerabilities. Based on these findings researchers could perform a deeper investigation of these unique characteristics, and how they could be leveraged to improve detection accuracy. Furthermore, practitioners are encouraged to avoid using VDMs trained on contemporary software directly for vulnerability detection in AI/ML software. In addition, small-moderate adaptation can greatly improve the detection accuracy, so, if practitioners have such instances, they should consider fine-tuning these models.

V. Conclusion

In this paper, we examined whether AI-based vulnerability detection models (VDMs) built on contemporary software can accurately generalize to AI/ML-specific software, and evaluated the extent to which domain adaptation can improve detection accuracy. We fine-tuned Transformer-based models (i.e., CodeBERT and CodeGPT) on contemporary data and evaluated them in a cross-domain setting. While the studied models achieved strong within-domain detection accuracy when trained and tested on contemporary software (around 92% F_1-Score for both models), we observed a substantial accuracy drop when employed on the AI/ML-specific subset (27% F_1-Score for CodeBERT and 31% for CodeGPT). Gradual adaptation of target-domain samples improved cross-domain F_1-Score up to 82%, with 10% augmentation for CodeGPT and 20% augmentation for CodeBERT already leading to more than 70% F_1-Score. In addition, UMAP visualization revealed a steady drift in learned representations and a progressively clearer separation between vulnerable and non-vulnerable AI/ML functions as more target-domain data were included. Finally, the explainability analysis using the self-attention mechanism showed an observable drift in the attention of the tokens between adaptation steps. Overall, the results indicate that VDMs built only on contemporary software cannot be reliably employed to AI/ML-specific software without proper domain adaptation, indicating that AI/ML-specific software exhibits distinct characteristics. However, even a small-moderate amount of target-domain data may suffice for satisfactory cross-domain detection.

Several directions for future work can be identified. First, we plan to extend our analysis to additional model architectures as well as additional software security tasks (e.g., vulnerability repair) to assess whether the observed cross-domain behavior reflects a more general phenomenon. Second, we aim at constructing a larger dataset of AI/ML-specific vulnerabilities, potentially including multiple programming languages. Furthermore, we plan to complement the current qualitative analyses with a quantitative evaluation, such as intra-class and inter-class distance measures for representation drift and categorization of token types in the attention drift analysis.

Acknowledgment

Work reported in this paper has received funding from the European Union's Horizon Europe Research and Innovation Program through the SECASSURED project, Grant Number 101225858.

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] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, 2024.

[4] K. Filus and J. Domańska, "Software vulnerabilities in tensorflow-based deep learning applications," *Computers & Security*, p. 102948, 2023.

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

[6] Z. Sheng, Z. Chen, S. Gu, H. Huang, G. Gu, and J. Huang, "Llms in software security: A survey of vulnerability detection techniques and insights," *ACM Computing Surveys*, vol. 58, no. 5, pp. 1–35, 2025.

[7] N. Shiri Harzevili, A. Boaye Belle, J. Wang, S. Wang, Z. M. Jiang, and N. Nagappan, "A systematic literature review on automated software vulnerability detection using machine learning," *ACM Computing Surveys*, vol. 57, no. 3, 2024.

[8] Z. Feng, "Codebert: A pre-trained model for program-ming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[9] M. Siavvas, I. Kalouptsoglou, E. Gelenbe, D. Kehagias, and D. Tzovaras, "Transforming the field of vulnerability prediction: Are large language models the key?" in *32nd MASCOTS International Conference*, 2024.

[10] M. Siavvas, E. Gelenbe, and D. Kehagias, "Static analysis-based approaches for secure software development," *Security in Computer and Information Sciences*, p. 142, 2018.

[11] X. Zhou, S. Cao, X. Sun, and D. Lo, "Large language model for vulnerability detection and repair: Literature review and the road ahead," *ACM Transactions on Software Engineering and Methodology*, 2025.

[12] Z. Cai, Y. Cai, X. Chen, G. Lu, W. Pei, and J. Zhao, "Csvd-tf: Cross-project software vulnerability detection with tradaboost by fusing expert metrics and semantic metrics," *Journal of Systems and Software*, 2024.

[13] C. Zhang, B. Liu, Y. Xin, and L. Yao, "Cpvd: Cross project vulnerability detection based on graph attention network and domain adaptation," *IEEE Transactions on Software Engineering*, pp. 4152–4168, 2023.

[14] V. Nguyen, T. Le, C. Tantithamthavorn, J. Grundy, and D. Phung, "Deep domain adaptation with max-margin principle for cross-project imbalanced software vulnerability detection," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 6, pp. 1–34, 2024.

[15] I. Kalouptsoglou, M. Siavvas, A. Ampatzoglou, D. Kehagias, and A. Chatzigeorgiou, "Vulnerability prediction using pre-trained models: An empirical evaluation," in *32nd MASCOTS*. IEEE, 2024.

[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] 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:1907.11692*, 2019.

[18] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022.

[19] I. Kalouptsoglou, M. Siavvas, A. Ampatzoglou, D. Kehagias, and A. Chatzigeorgiou, "Transfer learning for software vulnerability prediction using transformer models," *Journal of Syst. and Software*, 2025.

[20] 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.

[21] I. Kalouptsoglou, M. Siavvas, A. Ampatzoglou, D. Kehagias, and A. Chatzigeorgiou, "Locvul: Line-level vulnerability localization based on a sequence-to-sequence approach," *Information and Software Technology*, vol. 189, p. 107940, 2026.

[22] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.