Thesis Proposal: Identification and Prevention of Adversarial Attacks in Recommender Systems

Muhammad Umar Zeshan

September 2023

1 Introduction

Artificial in today's age is a vast and the most trendy topic in the world given its importance. Recommender systems are also using AI methods nowadays. To have a better understanding of the topic of the research, the importance of recommender systems, especially in the field of software engineering needed to be understood.

1.1 Recommender Systems

Recommender systems have become essential tools that help consumers navigate the broad digital world at a time marked by an overwhelming number of information and options. These systems, which are driven by sophisticated algorithms and user data analysis, are crucial in determining how we perceive the Internet. Recommender systems are at the core of tailored user experiences, from e-commerce giants like Amazon to content streaming platforms like Netflix and the social media networks we use every day. This essay examines the fundamental significance of recommender systems in a variety of fields and how they have changed how we find, interact with, and consume material and goods. The capacity of recommender systems to offer individualized recommendations is one of the main reasons for their rise to prominence. These systems curate content, goods, and services for each user by looking at their behavior, preferences, and past data. By providing pertinent suggestions, reducing time spent searching, and establishing a stronger bond between users and the platform, personalization improves the user experience. Users are more likely to interact with and revisit platforms that are aware of their preferences, resulting in a positive feedback cycle. Not only do recommender systems improve the initial user experience, but they also greatly increase user engagement and retention. Users are more inclined to stay on the platform longer when they are regularly exposed to information or goods that match their interests. Longer sessions result in greater user happiness, which raises user retention rates—a crucial statistic for online businesses—and leads to improved user experience overall.

Recommender systems have proven successful in e-commerce by increasing conversion rates and generating income. These systems raise the possibility of earning a sale by recommending goods that fit a user's preferences or enhance their present selection. The average transaction value is also raised by utilizing cross-selling and upselling opportunities. Businesses that successfully use recommender systems frequently experience a significant improvement in their financial performance. In order to keep viewers interested and entertained, content providers and streaming platforms mainly rely on recommender algorithms. Users might easily get lost in the sea of possibilities due to the large libraries of films, TV series, music, articles, and more. Users are assisted by recommender systems in finding pertinent content they might not have found on their own. User happiness is boosted by this dynamic content discovery, which also stimulates exploration and more varied consumption patterns. Information overload has emerged as a critical concern as the digital world keeps growing. By selecting and recommending the most pertinent content, recommender systems help users by lessening their cognitive load. These systems make it easier and more fun to consume information by providing a carefully curated list of options. Using recommender systems, persons with similar interests can connect via social media and online communities. They encourage a feeling of community by suggesting people, organizations, or content that corresponds with a user's social network activities. In addition to keeping visitors interested, this interaction encourages user-generated material and conversations. The cornerstone of our digital existence, recommender systems now shape how we shop, consume content, and interact with people online. They are extremely important because they may tailor experiences, encourage engagement, increase income, and reduce information overload. Recommender systems work as essential guides in a time when there are many options and a flood of digital content, boosting our online interactions and helping organizations and platforms succeed. Recommender systems will remain a pillar of the digital experience for years to come, influencing how we navigate the digital world as they develop and adapt to changing user needs.

1.2 Recommender Systems in Software Engineering (RSSE)

The pace of technological development and the amount of information are both daunting in the field of software engineering (SE). The steady influx of code, documentation, tools, and best practices is a challenge for software engineers. Recommender systems have become essential tools in this setting, utilizing data analytics, machine learning, and artificial intelligence to give software engineers, testers, and project managers individualized and context-aware advice. This essay examines the crucial function that recommender systems play in SE, as well as their uses, advantages, and difficulties, as well as the revolutionary potential they have for the future of software development. Recommender systems, as described in the above section, also known as recommendation systems or recommendation engines, are an area of data science and artificial intelligence that aims to help users choose from a wide range of options. To create individual-

ized recommendations, these systems examine user behavior, preferences, and previous data. These suggestions in SE may pertain to a variety of areas, such as tool selection, issue tracking, documentation, code, and documentation.

The role of Recommender systems is very evident in a lot of software applications. Developers frequently run into problems while looking for pertinent code snippets, libraries, or functions during the development phase. By recommending code segments that fit the present coding context, recommender systems lessen this strain. These solutions speed up development and improve code quality, encouraging more effective and error-free programming. Another application is, the elimination of bugs, which is a crucial component of software development. Recommender systems aid with this process by suggesting potential fixes, previously reported problems, or knowledgeable developers who can offer advice. They also help to prioritize bug reports, making sure that urgent problems are resolved right away. In SE, lifelong learning is crucial. In order to help developers learn new skills and stay current with the newest technologies and best practices, recommender systems provide recommendations for pertinent documentation, courses, and online resources. The skill development of software engineering teams is considerably aided by this feature of recommendation systems. At the heart of collaborative software development are version control systems. Recommender systems can make recommendations for code reviewers, team members, or relevant branches, enhancing cooperation procedures and code integration. They aid in streamlining software development's frequently complicated and collaborative nature. There are many tools, plugins, and customizations available in the software development environment. The workflow of a developer is analyzed by recommender systems, which then provide tools or configurations that increase productivity. These suggestions make sure that developers operate in a setting that suits their individual requirements.

The use of Recommender systems in Software engineering is advantageous in many ways. Recommender systems greatly increase developer productivity by automating and improving various software development processes, freeing them up to concentrate more on original and challenging problem-solving. Code segment recommendations, best practices, and bug fixes all help to lower errors, which raises the caliber of the program. By recommending pertinent documents and learning tools, recommender systems promote knowledge transfer and shorten the learning curve for new team members. These solutions promote collaboration among software development teams through version control and collaborative advice, creating more effective and pleasant work environments. Recommender systems in SE will improve along with technology as it advances, opening up new possibilities for intelligent software development help. Recommender systems act as trusted allies, enabling software professionals to negotiate the complexity of SE with accuracy and efficiency, thereby influencing the future of software engineering in an era where information is abundant but time is limited.

1.3 Challenges in research in RSSE

The objective of research can be defined in any field by the challenges present in that field. Although in the past 5-6 years the trend of performing research activities in RSSE has increased, it still remains a challenging task. The interaction of software engineering methods and recommender system methodologies creates a special set of problems for research in recommender systems in software engineering (RSSE). There could be many reasons for the research difficulty, the vast array of data sources that RSSE must manage includes code repositories, issue tracking programs, version control data, documentation, and user reviews. It is difficult to integrate and interpret this varied data. Data in software engineering is frequently noisy and sparse. For instance, there can be little historical information available for some projects or poor-quality bug reports. These restrictions need to be accommodated by recommender systems. The data used in software engineering is often confidential and proprietary. When using this data for recommender systems, researchers must overcome privacy and security problems. When there is insufficient historical data to offer insightful recommendations, such as for new projects or new members of a development team, the cold start problem in RSSE arises. The solution to this issue is really difficult. Software development is a dynamic process that involves code modifications, problem patches, and changing specifications. To make recommendations that are pertinent, RSSE needs to adjust to these changes in real time. Massive volumes of data are produced by large-scale software initiatives. It is a significant problem to develop recommender systems that can grow to effectively manage this volume of data. It can be difficult to define suitable measurements for RSSE. The efficiency of recommendations in a software engineering context may not be fully captured by conventional recommendation metrics. In order to provide recommendations that are truly effective, software engineers frequently need to have a thorough awareness of the project goals, developer skill set, and task specifications. It is not simple to incorporate and use this context in RSSE. As a result of the necessity for researchers in RSSE to have a solid grasp of both software engineering procedures and recommender system methods, RSSE is an interdisciplinary field that necessitates proficiency in numerous domains. The effectiveness of RSSE depends on user adoption and acceptance as well as the accuracy of the recommendations. One difficult component of RSSE research is making sure that developers believe and implement the suggestions. New tools and techniques are continually emerging, causing the ecosystem of software development tools to change. RSSE must adjust to this changing environment. Developers frequently need to know why recommendations are made, hence it is crucial in SE to provide reasons. It can be difficult to include explainability in RSSE. To ensure that recommendations are not discriminatory or serve to perpetuate preexisting biases in software engineering methods, RSSE must address concerns of bias and fairness.

Due to the dynamic and diverse nature of software engineering data, the requirement for reliable and context-aware suggestions, privacy and security issues, and the continual development of software engineering techniques and

tools, the research in RSSE can be challenging. In order to create recommender systems that are efficient and can improve the processes and results of software engineering, researchers in this subject must overcome several obstacles.

1.4 Adversarial Attacks in RSSE

In Recommender Systems in Software Engineering (RSSE), adversarial attacks refer to conscious attempts to trick or manipulate recommender systems in order to produce recommendations that are unfavorable or destructive. These attacks may result in poor choices, coding flaws, or other undesirable outcomes, which can have serious effects on software engineering. There are so many hidden or clear motivations for the attackers in this regard. Attackers may try to increase the prominence of their own contributions or initiatives by manipulating suggestions. Disgruntled team members or outside parties may attempt to sabotage a project by undermining recommendations, which could result in subpar code or unstable projects.

Several recommender systems for software engineering (RSSE) have been developed in recent years to aid developers in their work and, perhaps, lessen the growing information overload brought on by the availability of data from numerous sources. The learning materials of the recommenders could be vulnerable to malicious attacks because these sources are open to alterations and contributions from the general public. In other words, it is possible to trick software recommender systems by taking advantage of these sources. Adversarial attempts produce perturbations to trick and confuse systems by breaking them down, impairing their ability to provide recommendations. For instance, an adversarial attack on recommender systems may support or disparage a product, depending on the intent, which would have a detrimental effect on the final recommendations. Similarly, malicious users may expose recommender systems to hazardous artifacts by altering training data that is accessible through OSS platforms. Software system disruptions may arise if a recommender is trained to deliver harmful outcomes based on Adversaries. For instance, a recent study reveals that there have been attempts to force Android apps to open ports covertly, enabling unwanted access. Security concerns in machine learning systems and all-purpose recommender systems are investigated via research on adversarial machine learning (AML). AML has so far been studied across a variety of fields, including online systems and image classification, and it addresses both threats and countermeasures. To the best of our knowledge, software engineering hasn't yet looked into the AML problem in recommender systems. In this article, we present one of the first empirical studies exploring the applicability and consequences of AML in RSSE. First, we demonstrate that no effort has been made to examine the problem of AML in RSSE using a literature survey of prestigious software engineering venues.

2 Literature Review

This section provides an overview of the literature concerned with adversarial attacks in Software Engineering. In recent years, a lot of research has been carried out to address the issues of adversarial attacks on software recommender systems. As the challenge is getting bigger, the need to conduct an efficient survey that can provide a basic understanding to the researchers of research being already done is also getting bigger. Concerning the detection of malicious code in machine learning, most of the studies are focused on the way of feature extraction. For instance, Shabatai et al. [**?**] provided the taxonomy for malware code detection using ML with a focus on feature extraction methods. They also covered the literature regarding attacks in terms of feature selection techniques (Grain Ratio, Fisher Score, Document Frequency) and Classification algorithms (Artificial Neural Networks, Bayesian Networks, Naïve Bayes, K-NearestNeighbor, etc.).

Bazrfishan et al. [2] name three primary techniques for finding malicious software: (1) signature-based techniques, (2) heuristic-based techniques, and (3) behavior-based techniques. They also look into a few aspects of malware detection and talk about how malware hides itself to avoid detection. However, neither the dynamic nor the hybrid techniques are taken into account in the aforementioned research. Similarly, a survey of malware detection techniques is presented by Souri et al. [30] and is classified into two categories: (1) signature-based methods and (2) behavior-based methods. The survey, however, does not offer an assessment of the most recent deep learning algorithms or a taxonomy of the different feature types employed in data mining techniques for malware detection and classification.

Ucci et al. [33] divide approaches into three categories based on (i) the goal job they aim to solve, (ii) the feature types collected from Portable Executable files (PEs), and (iii) the machine learning algorithms they employ. Although the study fully describes the feature taxonomy, it does not include recent research advances, particularly those involving deep learning and multimodal techniques. Ye et al. [38] discuss conventional machine learning methods for detecting malware, which includes processes for feature extraction, feature selection, and classification. However, crucial components like a file's entropy or structural entropy and some dynamic components like network activity, opcode traces, and API traces are absent. Additionally, recent hot areas for malware detection, such as deep learning techniques and multimodal approaches, are not explored. An investigation of malware's bibliometrics is given by Razak et al. [1]. It examines malware-related articles by nation, organization, or authors. However, the report cannot be regarded as state-of-the-art because it does not describe the features used by malware detection. Lastly, T. Nguyaen et al. [24] pointed out that hostile users can inject malicious code into Github to negatively impact the API recommendation systems, which are trained on insecure datasets. They also represented the potential risks to notable RSSE for mining APIs and/or code snippets.

By providing a framework for poisoning and evasion attacks, Demontis et al.

[8] described the transferability in adversarial attacks and demonstrated that transferability depends on the size of the input gradient (for the target classifier) and the loss variance (for the base classifier). Robustness is generally higher for simpler or regularized models that result in smaller input gradients. Another well-liked approach adopted by researchers to increase the resilience of models against adversarial evasion attacks is ensemble learning. This method combines several classifiers to increase the robustness of the classifiers. Y. Huang et al. [15] described that a DL model is prone to adversarial inputs, which are produced by introducing specific noise to regular inputs and cause prediction mistakes if it has this vulnerability. Zhang et al. [39] generate adversarial examples by renaming identifiers using Metropolis-Hastings sampling.

In 2019, Wang et al. [36] proposed a "sensitivity" metric to discriminate between neutral and hostile samples. they further presented a method to identify the adversarial input samples produced at runtime by SOTA attacking techniques. The basic technique involved was DNN mutation on the MINIST and CIFAR10 datasets to test the theory of adversarial samples.

Zhang et al.[42] presented the GA-based automated test technique to generate data with more diverse uncertainty patterns to contribute to distinguishing between AEs (Adversarial Examples) and Bes (Benign Examples) allowing for an empirical study on uncertainty metrics, organizing the current data into categories by carrying out the research to comprehend the traits of Bes and AEs produced by adversarial tools and attacks, and investigated the value of test data generation. Moreover, their outcomes showed how the generated data could be used to go beyond defense mechanisms. Ghamizi et al [13] presented a search-based approach that produces useful adversarial examples for applications involving random forests. they further examined how adversarial examples are produced when domain constraints are satisfied. On the real-world system of a major bank, they tested CoEvA2's capacity to construct legitimate hostile cases in order to assist in training the system in anticipation of fending off adversarial attacks. They investigated the challenge of evaluating an industrial credit rating system based on machine learning against fraudulent inputs. They specifically took into account the scenario in which an attacker modifies relevant features in an effort to deceive a binary classification machine. To this purpose, they assessed the most cutting-edge adversarial assaults utilizing our partner's dataset and system, both in a full-knowledge environment and a limited-knowledge one.

On MINIST, CIFAR10, and ImageNet, the Black-box Momentum Iterative Fast Gradient Sign Method (BMI-FGSM) was tested by Lin et al. [21] in 2020. They put forth the BMI-FGSM, a brand-new black-box adversarial sample generation technique. Model architecture, weights, or gradient understanding are not necessary for this strategy in their final outcomes. They also compared the model's findings to those of other cutting-edge techniques like the white-box MI-GSM method and the black-box ZOO approach. Results indicated that this method can cause misclassification to occur for a certain target output label. In 2021. Huang et al. [16] studied the TF lite DL models and the relations between them, especially in terms of adversarial attacks on the deployment of

DL models in Android apps. By locating highly comparable pre-trained models from TensorFlow Hub, they put forth a workable strategy for adversarial attacks on deep learning models. Experimental results show that attacks with accurate information from pre-trained models outperform blind attacks by a factor of more than three. The results also demonstrate that when the fine-tuned model and its pre-training model become more similar, the robustness of the fine-tuned model declines. In their empirical investigation of the application of deep learning models in actual Android apps, they found 267 pre-trained TFLite models spread throughout 62,822 Android applications. After removing 182 identical models, they discovered that 61.18 percent of the remaining models share a structural similarity of more than 80 percent with pre-trained models from TensorFlow Hub. However, this research effort was more concerned with Android apps and was not highly related to adversarial attacks in software engineering.

Shriver et al. [28] provided a method for minimizing DNN correctness issues in order to make falsifier applications, particularly adversarial attacks, easier. They also conducted an empirical study to evaluate the cost-effectiveness of property reduction, the scalability of property reduction, and the cost-effectiveness of property falsification in detecting property infractions. They used the method of defining property reduction and reduction transformation in great effect. Zhao et al. [45] applied K-NN and Z score detection approaches to classify and characterize the adversarial examples. The distinction between Adversarial and Benign Examples was also proposed, and it was noted that AEs are less robust than BEs. They further presented a defense framework called attack as defense (A2D) to find AEs by assessing how robust an example is. They suggested a novel categorization that uses robustness to discriminate between adversarial and benign situations and offers characterization-based detection methods that can make use of current attacks and do not require changing or retraining the protected model. Moreover, they undertake in-depth tests to validate our findings and demonstrate that our protection beats recently released, promising detection techniques. They carefully examine potential adaptive attacks against our defense and assess them against it while integrating a complementary detection strategy and adversarial training. The integrated defense has a lot of potential.

In 2021, Zhang et al [41] focused on the Backdoor type of adversarial attacks. In this regard, they presented an Adversarial Backdoor, that can successfully insert a human-imperceptible backdoor into DL systems and perplex several existing detection techniques. They used two cutting-edge adversarial attack techniques to implement the hostile backdoor assault. The source code is accessible to everyone. Finally, they carried out comprehensive tests to show the viability and adaptability of Adversarial Backdoor. Targeted Universal Adversarial Perturbation (TUAP) is the trigger for the Adversarial Backdoor attack that they suggest. The distribution of training data can be used by Adversarial Backdoor to reduce anomalies and perplex current detection techniques. To create an adversarial backdoor, they leverage data poison to create a poisoned model and generate TUAP by converting current adversarial attack techniques into TUAP creation algorithms.

Model Mutation Testing (MMT) was utilized by Chen et al. [5] as a baseline, and Graph Guided Testing (GGT) was applied to develop it in order to increase the variety and decrease the number of parameters used in MMT. We suggest a technique for drastically lowering the computational burden of DNN models called graph-guided pruning. When there is no accuracy loss and the average degree, the graph's (AD) is utilized to regulate the number of FLOPs in the DNN model that was created. They further suggest Graph-Guided Testing (GGT), which can identify adversarial samples more effectively, for adversarial detection is a great deal where there is less computing effort and models, contrary to using MMT. They discover a correlation between the Average Shortest Path Length (ASPL) and the performance of GGT's adversarial detection as well as the accuracy of individual models. The graph with shorter ASPL may naturally produce a DNN model with greater accuracy, which can then be utilized to better differentiate between adversarial samples and typical samples, making it more appropriate to create their GGT. On CIFAR10 and SVHN, we compare our GGT with MMT, and we discover that GGT performs better than MMT in terms of accuracy and efficiency. While GGT detects 93.61 percent of adversarial samples on CIFAR10 and 94.46 percent on SVHN with only 30.04 and 28.50 pruned models on average, MMT uses an average of 51.12 and 77.16 models to achieve an average accuracy of 74.74 percent and 44.79 percent on CIFAR10 and SVHN, respectively. Additionally, the FLOPs of a single model used in GGT are only 5.22 percent of those used in MMT.

Zhang et al. [40] addressed the shortcomings of prior methods that lack portability and only provide adversarial examples for a specific DNN model. Subsequently, They proposed CAGFuzz, which can produce AEs for common DNN models in three steps: 1) AEG training to address the transferability issue 2) Removing deep features to limit the adversarial examples 3) AEs are used to keep the models. They create AEG, an adversarial example generator, which can produce adversarial examples based on common data sets with little perturbations. They take the deep features from both the original and the adversarial examples and use similarity measurement to bring them as close to one another as they can. Finally, based on a variety of open data sets, they create a number of tests to assess the CAGFuzz method. Coverage-guided adversarial generative fuzzing testing is known as CAGFuzz. For a given dataset, CAGFuzz develops an adversarial example generator. It iteratively takes original instances, generates adversarial examples, generates feedback on the coverage rate, and finds potential flaws in the DNN's development and deployment phases. In order to effectively construct adversarial inputs, Gou et al. [14] presented a novel search algorithm that is based on the internal workings of RNNs and optimizes the irregularity of RNN states. The study then advanced two state-based RNN-specific coverage metrics, primarily as recommendations for adversarial testing. They first show that, in contrast to other techniques, coverage guidance has a diversified searching capability for adversarial inputs. In RNN testing, they demonstrated that their state coverage guidance is also superior to neuron coverage guidance, and finally, they examined the reliability and efficacy of RNN-Test.

Islam et al. [17] studied the bug-fix patterns in DL applications and presented a comprehensive study in this regard. First off, they discover that DNN's bug-fix patterns deviate dramatically from conventional bug-fix patterns. Second, their findings indicate that developers of DNN software can benefit significantly from addressing the incompatibility between the data and the DNN on their own, particularly if they are made aware of the potential effects of their bug fixes on the resilience of the DNN model. Third, their research reveals that version upgrades are a typical problem-fix strategy. Although SE research has extensively explored version upgrades, their bug fix patterns indicate that automated repair tools will need to handle at least two distinct problems: intrusive, backward incompatible changes and probabilistic behavior change. Although that was quite an interesting and advanced level of findings and research topic, it was not directly related to the adversarial attacks in software engineering. In 2019, X Du et al. [10] presented DeepStellar, a general-purpose quantitative analysis framework that makes it possible to carry out efficient security and quality studies for RNN-based DL systems. DeepStellar's core is an abstraction of the distinct hidden state space of RNNs, which represents the internal behaviors of RNNs as a discrete-time Markov chain (DTMC). They created powerful adversarial sample detectors for RNNs and an efficient test generation tool based on DeepStellar to demonstrate the power of the software. With hundreds of times more adversarial samples created and 97 percent accuracy in adversarial attack detection, they deploy these tools to three genuine RNN models, including ASR and image classification, and see promising results. But again, this research effort was not as relevant to the cause of AML in software engineering. X Zhang et al. [43] offered a trained neural network instance-based visualization method. This method can show both the static organization and the changing behavior of neural network models. They create user interfaces that let programmers communicate with the visible neural network. These interfaces make it easier for developers to change the model's structure and thus analyze its behaviors. Finally, they implemented their approach into a tool, namely NeuralVis, to validate our approaches. NeuralVis offers engineers an interface to compare the intermediate outputs of two inputs, analyze the activation status of a chosen layer, and mutate the model by freezing filters in convolutional neural networks(CNN).

Kwiatkowska et al. [18] in their lecture, explains the adversarial search strategy in relation to white box and black box strategies. They admitted that many issues still exist despite the fact that the approaches that have been described have been helpful in quantifying the local and global adversarial robustness of deep neural networks and Bayesian neural networks. Diddee et al. [9] in their short paper, suggested CrossPriv, a model for cross-silo federated learning systems that protect user privacy, in order to establish some initial standards for collaborative software that is SaaS-based. They simulated the model's effectiveness by demonstrating a medical use case in order to further highlight the advantages of the suggested model for a collection of healthcare facilities aiming to accomplish a similar objective, in this case, to develop a robust and diverse deep learning model to detect pneumonia using X-rays. In order to comprehend

the data distribution and its impact on DL testing operations, Bernard et al. [3] carry out a substantial empirical investigation using the most recent out-of-distribution OOD approaches. Their findings demonstrate that, even in difficult situations when the test data is strikingly similar to the training data, the OOD detection approaches now in use can separate the OOD data from the recently generated test cases. Their analysis also demonstrates how the distribution of the generated test cases can be significantly impacted by the testing requirements and image mutation operators currently in use. Finally, they show that distribution-aware datasets are typically more effective at enhancing resilience. The research effort was quite good in both above papers, but again, it was not entirely related to any kind of adversarial threat or malware detection. Sun et al. [31] showed that they created the first DNN concolic testing technique. They assess the approach using a variety of test coverage criteria, such as Lipschitz continuity and various structural coverage measures. We use experiments to demonstrate how well our new method handles this wide variety of features. In the software application DeepConcolic, they use the concolic testing methodology. According to experimental findings, DeepConcolic obtains great coverage and is able to find a sizable number of hostile examples.

In 2020, To arrange the connected code according to its authors, which has never been explored in the earlier study on Android, wang et al. [35] suggested authorship decoupling in the granularity of the package. Based on this, they discovered that a number of classes, such as configuration classes and third-party libraries, were included during the compilation of the APK file. They take three types of features—dex-level, lib-level, and manifest-level—from the main module in order to identify authorship. Instead of using the Android framework to extract features, they mix TF-IDF and word2vec approaches to embed the features. The identification of authorship is then accomplished using three classifiers. Their method improves authorship identification accuracy by 3.4 percent when compared to an open-source authorship attribution tool. They addressed the issue of adversaries in Android applications, but in general, they did not highlight the threat of adversarial attacks in software engineering.

S. Yan et al. [37] find in their research that the relationship between DNN coverage criterion and model quality (as determined by model accuracy in the presence of adversarial samples) is not monotonic. Though finding adversarial examples can be aided by using DNN coverage requirements, successful adversarial examples may not result in more coverage. Comparatively to existing gradient-based algorithms, existing methods used to produce adversarial instances based on coverage requirements typically include greater and human observable disturbances. Such inputs can be compared to program inputs that might not adhere to the software input requirements when used in model testing. A model can be retrained using adversarial examples produced via DNN coverage-guided approaches to increase model robustness against the adversarial method used to generate the training inputs (for example, adjusting blurriness to maximize coverage while maintaining semantics). However, these models are not resistant to assaults that use gradients (such as PGD). On the other hand, PGD-based adversarial training can increase the model's resistance to

PGD attacks, but not to attacks that use coverage as their primary source of information. Zhang et al. [44] explored and studied the idea of dynamic DNN slicing. They introduce NNSlicer, a dynamic slicing method for DNNs based on neural network data flow analysis. A neural network slice is defined as a subset of neurons in the neural network that exhibits greater effects on the slicing criterion, whereas the slicing criterion is defined as a set of neurons with special meanings (such as the neurons in the final layer of an image-classification model whose outputs represent the probabilities of categories). Instead of static slicing, which is input-independent, NNSlicer focuses on dynamic slicing, where a slice corresponds to a set of input samples. They create three intriguing applications with DNN-slicing methods and show how useful NNSlicer is.

In 2020, Compton et al. [6] presented the idea of using the recently introduced embedding approach code2vec to study the effects of adversaries on its training. They showed an analysis of variable name obfuscation during code2vec embedding model training and testing. They also contributed to an analysis of code2vec embeddings for a variety of tasks requiring source code classification. They find that by using basic aggregate functions on the individual method embeddings, baselines can be used to create embeddings for files containing numerous methods. Brandon et al. [4] provided a methodology for root cause analysis for service-oriented architectures based on graphs. In order to determine the reason for an abnormal graph representation of the system, they describe a graph comparison engine that enables them to compare it to an earlier instance. They develop a system that can create a graph representation of the architecture state using a collection of monitoring technologies. Three service-based architectures that are frequently used in industry are deployed using containers as part of a series of real-world tests in which we evaluate the RCA process. The hosts and containers are then given anomalies to deal with. The findings demonstrate how, compared to other approaches that ignore these factors, their technique is able to gather more information about the variables that may affect or be affected by the anomaly. That study was quite important in the anomaly detection type of adversarial attacks. Sauvanaud et al. [27] contributed to the definition of a new ADS for cloud services that allows for the detection of two sorts of anomalies (errors and SLA violations (SLAV)) and offers the cloud provider two degrees of diagnosis (i.e., locating the anomalous VM and identifying the kind of error that caused the anomaly). They further showed the deployment and validation of their ADS on a cloud computing platform powered by VMware, along with thorough sensitivity tests that highlight the effectiveness of its detection capabilities when employing supervised machine learning algorithms. The IP multimedia subsystem (IMS), which was created as a Virtual Network Function (VNF), and the MongoDB database are the two case studies used for the validation. J Singh et al. [29] presented a Three-step Malware Classifier for ML Algorithms Using text mining techniques, readable strings are analyzed word by word to create a very high-dimensional string feature matrix. Secondly, in order to account for the randomness of API and PSI features, Shannon entropy is estimated across printed strings and API calls. Finally, in order to create malware classifiers using machine learning techniques,

all features are finally incorporated into the training feature set.

In 2020, Fidalgo et al. [11] analyzed the vulnerability risk for PHP and SQLi apps. A dataset of PHP opcode slices, pre-trained embedding vectors for each PHP opcode, an experimental evaluation offering various assessments of various hyperparameter configurations, the analysis of PHP web applications in the intermediate PHP opcode language, a DL model that accurately categorizes PHP opcode slices as SQLi vulnerable or non-vulnerable, and a dataset of PHP opcode slices are all included in their contributions in this research. However, the article does not focus on a specific type of malware or adversary in terms of software engineering. Monni et al. [23] proposed EmBeD, an energy-based approach to anomaly detection that makes use of the free energy calculated with a Restricted Boltzmann Machine on KPIs to find anomalies in cloud systems that are prone to failure. The EmBeD can reliably and timely anticipate failures with a low percentage of false positives and very little runtime cost, according to the experimental results described in the paper. It does not require training with seeded errors, which is why, it eliminates the primary drawbacks of existing statistical and machine learning-based methods.

In 2021, Usman et al. [34] proposed NEUROSPF, which is a program that makes it easier to analyze neural networks symbolically. It can handle feed-forward neural networks with dense, convolutional, and pooling layers, as well as ReLU activations and Softmax functions. They tested NEUROSPF on three neural networks (CIFAR10, MNIST low quality, and MNIST high quality), demonstrating its capacity to handle intricate neural network models and emphasizing the value of employing peer classes. Y Li et al. [20] begin their investigation into strengthening Deep Learning DL-based code authorship attribution techniques. They choose to concentrate on this family of technologies because they are particularly promising for real-world adoption and can automatically acquire coding style trends (i.e., avoiding the time-consuming involvement of domain experts). They suggest a ground-breaking system called Robust Coding Style Patterns Generation (RoPGen), which basically teaches authors their own coding style patterns that are challenging for attackers to modify or replicate. The main concept is to integrate gradient and data augmentation during the adversarial training stage. As a result, the diversity of training samples is successfully increased, deep neural network gradients are meaningfully perturbed, and diverse representations of coding styles are learned. V. Nguyen et al. [25] approached vulnerability detection as an inductive text classification problem and provided ReGVD, a straightforward but powerful graph neural network-based model for vulnerability detection in source code. Each raw source code is converted into a graph by ReGVD, which initializes the node feature vectors using only the token embedding layer of the pre-trained programming language model. ReGVD then uses a combination of the sum and max poolings and residual connections between GNN layers to learn graph representation. To assess and contrast the signal awareness of AI-for-code models, Suneja et al. [32] introduced the P2IM prediction-preserving input minimization method. P2IM, in particular, methodically condenses a program sample to the smallest possible fragment that a model needs to reach and maintain its first susceptible predic-

tion. P2IM examines the model's dependency on false signals by examining if the minimal snippet has the same vulnerability as the original sample. They use Signal-aware Recall (SAR), a new metric we propose, to quantify the signal awareness of three cutting-edge neural network models utilizing P2IM across several datasets.

All this relevant literature review, one thing is evident the potential of the research in recommendation systems is huge and has not been addressed properly, specifically in terms of malware detection in software systems.

3 Research Methodology

The research methodology works in 2 steps: 1. Review of the available literature to date in the field of adversarial attacks in software engineering. 2. Identifying and defining the type of attacks to build a comprehensive malware detection model.

3.1 Overview of the Literature

The systematic mapping studies (SMS) conducted in this work adhere to the recommendations made by Petersen et al []. The suitability of the criteria and the approach were evaluated by a pilot study on a small sample of articles, which sparked discussions and revisions on the procedure. The many steps of the exploration phase are described in this section as they are laid out in the protocol's final form.

3.2 Research Questions

The purpose of this study is to analyze the research on adversarial machine learning for software engineering. In order to achieve this, we look at what has already been done, what might be regarded as common practice, and what has not yet been investigated.

RQ1: How well does the currently available literature address the problem of AML in Software Engineering? We conduct a literature review to see if there has already been any work done to understand and address threats to Software Engineering stemming from malicious data.

RQ2: What are the different kinds of threats that can impact software engineering systems? This question aims to identify the different types of attacks and the way they cause damage.

RQ3: How a proper state-of-the-art malware detection system can be developed that can handle the different types of adversarial attacks in software engineering?

3.2.1 Inclusion Criteria:

The inclusion criteria define the scope of our systematic mapping study and enable us to identify papers that will help build answers to the research questions.

We have focused on those papers that: 1. Are written in English, peer-reviewed, and published between 2010 and 2023. 2. Focus on one of these knowledge areas from the:

- Adversarial Machine Learning and Software Engineering.
- Adversarial Machine Learning and Recommender systems.

3.2.2 Exclusion Criteria:

Exclusion criteria enabled us to filter out irrelevant papers. Papers featuring one of the following characteristics were excluded:

- Does not provide assistance for at least one of the considered software engineering research tasks related to AML.
- Does not provide any information regarding the role of AML in recommender systems.
- Is a survey, a systematic literature review, or a mapping study?

3.2.3 Data Extraction

We took the following information from each article:

1. The authors, title, publishing year, and source (journal or conference). This information was taken straight from the editor's website.

2. The name or type of the attack/threat that was finally either mentioned in the paper's title or introduced in its entirety.

3. An outline of the authors' descriptions and objectives for the particular AML problem. 4. Whether or not the solution is a software engineering recommender system (RS), which makes recommendations.

5. The concept of a recommender system for software engineering is based on the definition adopted by Robillard and Walker in their book [] a software application that provides information items estimated to be valuable for a software engineering task in a given context.

6. Whether a user study had been carried out as stated in the paper's evaluation section.

3.2.4 Selected Papers

This section compiles data on the chosen papers. The number of papers on software assistants that were published between 2011 and 2023 is shown in Figure.

There was a specific query developed to search the papers which are relevant to the research topic. The query searches for keywords like "Adversarial, Malicious, Vulnerable, Software Engineering, Machine Learning" in the abstract and titles of the articles from all the famous SE venues. Most of the papers are selected from the years 2020 and 2021, which demonstrates that the research is in need of an hour and is trending. It also describes that there are no such dedicated venues for addressing adversarial attacks in software engineering which also provides extra motivation to do research on this topic.

3.3 Analysis and Classification Results

This section is devoted to answering our research questions.

3.3.1 RQ1: How well does the currently available literature address the problem of AML in Software Engineering?

We relied on each paper's author-provided description of the adversarial attacks to provide the answer to this research question. Extracting the most relevant research papers is dependent on the most suitable research query.

This query provided an extensive search of relevant papers involving adversarial attacks in both the Software Engineering and Recommender Systems domains. We learned about the adversarial attacks that each paper describes from these specifics, and based on the inclusion and exclusion criteria, we created our final list of papers that are excellent candidates for a primary review of information about adversarial threats. We also took into consideration less relevant papers if they made a significant contribution to this popular research field. There were about 130-135 papers that were selected using the query and then further filtration was done to determine whether that paper was relevant to the AML or not. The selected papers then was classified in terms of the type of defects they addressed. Based on this classification, the RQ2 was described.

3.3.2 RQ2: What are the different kinds of threats that can impact software engineering systems? This question aims to identify the different types of attacks and the way they cause damage

To answer this question, we need to define the different kinds of attacks mentioned in the articles that are under investigation in our research study.

As mentioned above, most of the relevant studies are concerned with a limited number of prominent adversarial attacks.

1. Backdoor Attacks

Kwon et al. [19] discussed the issue of backdoor attacks in deep neural networks DNNs damaging the classification results of the networks. Attackers are able to actively acquire DNN training data through backdoor assaults, which they can use to train extra harmful data, such as the precise trigger. In normal circumstances, DNNs categorize normal data accurately, however malicious material with a particular trigger learned by attackers can lead to DNN misclassification. Further, they elaborated on the positive role of backdoor attacks as well, as they can be used to create confusion for the enemy in war situations. Although they highlighted the backdoor adversarial attacks well, still they are only concerned about their effects in terms of DNNs without presenting the complete picture in terms of the consequences of Backdoor attacks in software engineering.

Rohan et al. [22] also elaborated on the effects of backdoor attacks in machine learning models. They mentioned that despite not having access to the model parameters or the input data used to train the model, black box attacks

can still be utilized to deceive a target model. Later they developed a defense mechanism to tackle the backdoor attacks.

2. Data Poisoning Attacks

Dang et al. [7] also highlighted the issue of vulnerability of DL methods in dealing with adversarial attacks. One of the most recent and dangerous attacks is the data poisoning attack and it is different than the typical adversarial attack. They highlighted the difference between adversarial and data poisoning attacks' by identifying the phase of the DL method in which the attack is occurring. Attacks that happen during a training phase are adversarial attacks (also known as evasion attacks), and attacks that happen during a test phase are poisoning attacks. The results of poisoning attacks are cautious for nervousness and incorrect viewpoints, discrimination in AI (but of course, it is not right due to the poisoned models or biased training), and more dangerous outcomes than evasion attacks, from the perspective of social influence. So, they introduced an effective real-time attack detection system which can not guarantee to evade the attacks 100 percent, but can point out the attacks in most of the cases. They concluded that their attack strategies and responses are still in their infancy and have the potential to lead to new developments in model verification, machine learning, and statistical learning theory.

3. Evasion Attacks

Evasion attacks mainly focus on manipulating certain features to confuse the ML classifiers via packet perturbations. Ganesan et al. [12] studied the impact of evasion attacks that cause network intrusion. They presented a defense mechanism against network intrusion, especially in the age of the internet named NIDS (network intrusion detection system). In evasion attacks, the adversary initiates an attack by tampering with the packet's features in an effort to get past the ML-based NIDS. The accuracy of the ML-based NIDS can be reduced from 99+ percent to 0 percent by the adversaries using packet forging techniques to cause an ML classifier to inaccurately label attack packets as benign. The adversary with domain expertise and access to packet crafting techniques can change packet headers and launch assaults on ML-based NIDS when we utilize ML classifiers in the normal manner, using all relevant characteristics in the dataset to train the model. They suggested using a number of smaller feature sets to find ML classifiers that, when used in an ensemble, would be more resistant to evasion attempts. Their practical assessments utilizing well-known datasets have demonstrated the effectiveness of the suggested ensemble classifier in recognizing many evasion methods that were missed by the conventional usage of a single ML classifier with the entire training dataset. Out of the literature that is under study, no other significant contribution is noticed which deals especially with evasion adversarial attacks which reiterates the potential of research and investigation in this area.

4. Random Forest Attacks

Salah et al. [13] shed light on the phenomena of random forest attacks. They take into account the technique put forth by Papernot et al. [26], hereafter known as the Papernot attack, which was initially intended to lead to misclassifications in decision trees by visiting every node in the tree and forc-

ing them to flip until the misclassification was achieved. They modify this adversarial approach (against random forests) by repeatedly applying the Papernot attack to each forest tree until the random forest's categorization result changes. This approach is known as Iterative Papernot. On all of the initial test inputs from their use case where the model correctly classifies data, they tested Iterative Papernot. They showed that although their results are quite an improvement in creating the attacks but overall success rate is almost zero because of the domain constraints at the input level. They also explored another family of adversarial attacks called Gradient-based attacks. A DNN iteratively modifies the weights of its neurons as it learns in response to the gradient of its cost function, which is dependent on the weights. The same information is used in gradient-based attacks to create a perturbation that alters the last neuron layer's output and, in turn, the classification result. In both cases, they observed that creating examples of these adversarial attacks is not very useful due to different domain constraints. So, they introduced a search-based method to overcome these problems, and at the same time, they improved the defense system by exploiting the attack mechanism.

We observed that all these types of adversarial attacks are not addressed properly in literature as only a few notable contributions are available which are mentioned while addressing RQ2.

3.3.3 RQ3: How a proper state-of-the-art malware detection system can be developed that can handle the different types of adversarial attacks in software engineering?

The malware detection system needs a great understanding of the state-of-the-art and literature background in this field and this RQ is still under study. However, the general state of the art is addressed below.

1. Deep learning and machine learning

Deep Learning Models: The problem of malware detection has been applied to Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models. These models are able to learn intricate behavioral and coding patterns. Traditional machine learning methods frequently call for properly designed features. For the purpose of detecting malware, researchers have been looking towards automating the extraction of pertinent aspects from code and behavior data.

2. Analysis of behavior

Dynamic analysis: Investigating a piece of software's behavior while it is in use can reveal details about what it is doing. Malicious activity is found via anomaly detection and behavior profiling. Static Analysis: Without running the software, the code and metadata can be examined to identify potential dangers. Code structures and patterns are frequently examined using static analysis techniques.

3. Graph-based Methods

The discovery of connections and interdependence between code components is made possible by representing software as graphs. GNNs, or graph neural

networks, have demonstrated potential in identifying these linkages.

4. Ensemble Techniques

The overall accuracy and robustness of malware detection systems can be increased by combining various machine learning models or analysis approaches into an ensemble.

5. NLP: Natural Language Processing

To analyze and find dangerous code patterns in textual descriptions, documentation, or comments in software repositories, several malware detection systems use NLP approaches.

6. Data enhancement and artificial data

The training of machine learning models can be enhanced and their ability to generalize to new threats increased by creating synthetic harmful samples and enhancing datasets with variants of well-known malware.

7. Adaptive Learning:

To make use of their feature extraction capabilities, pre-trained models from other fields, such as computer vision or natural language processing, can be adjusted for malware detection tasks.

8. Embedded and Internet of Things Malware Detection:

Malware detection in various fields has gained attention due to the rise of IoT devices and embedded systems. Techniques frequently entail examining device activity, network traffic, and firmware.

9. Detection of Adversarial Attacks:

Finding adversarial assaults on malware detection is essential as attackers get more skilled. To make malware detection systems resistant to evasion attacks, research has been done.

10. Clarity of Explanation and Interpretability:

Trust and debugging depend on knowing why a malware detection system takes a particular action. In this context, various approaches to explaining machine learning models' judgments have been investigated.

11. Scalable and real-time solutions

Real-time, scalable malware detection systems should be able to manage enormous amounts of software artifacts and upgrades.

12. Continual Education:

It is crucial to create systems that can adjust to changing program behaviors and virus threats.

These are basic understandings of the topics, which can be modified later to make an effective malware detection system.

4 Timeline

The project is divided into 2 parts: The literature background and theoretical part, which is almost complete and a paper is in the final stages of writing to submit in a relevant SE venue in the next month. The next part is to build a state-of-the-art malware detection system, which requires extensive experiments which are time-consuming. So, it can be anticipated that by the end of the

second year of PhD, this process can be completed. As in Ph.D., doing research achievements is an iterative process, so the steps to the final projects will be addressed properly and will be published accordingly in relevant venues.

References

[1] Mohd Faizal Ab Razak, Nor Badrul Anuar, Rosli Salleh, and Ahmad Firdaus. The rise of "malware": Bibliometric analysis of malware study. *Journal of Network and Computer Applications*, 75:58–76, 2016.

[2] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*, pages 113–120. IEEE, 2013.

[3] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. Cats are not fish: Deep learning testing calls for out-of-distribution awareness. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, pages 1041–1052, 2020.

[4] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software*, 159:110432, 2020.

[5] Zuohui Chen, Renxuan Wang, Jingyang Xiang, Yue Yu, Xin Xia, Shouling Ji, Qi Xuan, and Xiaoniu Yang. Detecting adversarial samples with graph-guided testing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1196–1198. IEEE, 2021.

[6] Rhys Compton, Eibe Frank, Panos Patros, and Abigail Koay. Embedding java classes with code2vec: Improvements from variable obfuscation. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 243–253, 2020.

[7] Tran Khanh Dang, Phat T Tran Truong, and Pi To Tran. Data poisoning attack on deep neural network and some defense methods. In *2020 International Conference on Advanced Computing and Applications (ACOMP)*, pages 15–22. IEEE, 2020.

[8] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium (USENIX security 19)*, pages 321–338, 2019.

[9] Harshita Diddee and Bhrigu Kansra. Crosspriv: user privacy preservation model for cross-silo federated software. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1370–1372, 2020.

[10] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. A quantitative analysis framework for recurrent neural network. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1062–1065. IEEE, 2019.

[11] Ana Fidalgo, Ibéria Medeiros, Paulo Antunes, and Nuno Neves. Towards a deep learning model for vulnerability detection on web application variants. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 465–476. IEEE, 2020.

[12] Aparna Ganesan and Kamil Sarac. Mitigating evasion attacks on machine learning based nids systems in sdn. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 268–272. IEEE, 2021.

[13] Salah Ghamizi, Maxime Cordy, Martin Gubri, Mike Papadakis, Andrey Boystov, Yves Le Traon, and Anne Goujon. Search-based adversarial testing and improvement of constrained credit scoring systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1089–1100, 2020.

[14] Jianmin Guo, Quan Zhang, Yue Zhao, Heyuan Shi, Yu Jiang, and Jiaguang Sun. Rnn-test: Towards adversarial testing for recurrent neural network systems. *IEEE Transactions on Software Engineering*, 48(10):4167–4180, 2021.

[15] Hanxun Huang, Yisen Wang, Sarah Erfani, Quanquan Gu, James Bailey, and Xingjun Ma. Exploring architectural ingredients of adversarially robust deep neural networks. *Advances in Neural Information Processing Systems*, 34:5545–5559, 2021.

[16] Yujin Huang, Han Hu, and Chunyang Chen. Robustness of on-device models: Adversarial attack to deep learning models on android apps. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 101–110. IEEE, 2021.

[17] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. Repairing deep neural networks: Fix patterns and challenges. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1135–1146, 2020.

[18] Marta Kwiatkowska. Safety and robustness for deep learning with provable guarantees. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–3, 2020.

[19] Hyun Kwon, Hyunsoo Yoon, and Ki-Woong Park. Friendnet backdoor: indentifying backdoor attack that is safe for friendly deep neural network. In *Proceedings of the 3rd International Conference on Software Engineering and Information Management*, pages 53–57, 2020.

[20] Zhen Li, Guenevere Chen, Chen Chen, Yayi Zou, and Shouhuai Xu. Ropgen: Towards robust code authorship attribution via automatic coding style transformation. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1906–1918, 2022.

[21] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Black-box adversarial sample generation based on differential evolution. *Journal of Systems and Software*, 170:110767, 2020.

[22] Rohan Reddy Mekala, Adam Porter, and Mikael Lindvall. Metamorphic filtering of black-box adversarial attacks on multi-network face recognition models. In *proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*, pages 410–417, 2020.

[23] Cristina Monni, Mauro Pezzè, and Gaetano Prisco. An rbm anomaly detector for the cloud. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 148–159. IEEE, 2019.

[24] Phuong T Nguyen, Claudio Di Sipio, Juri Di Rocco, Massimiliano Di Penta, and Davide Di Ruscio. Adversarial attacks to api recommender systems: Time to wake up and smell the coffee? In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 253–265. IEEE, 2021.

[25] Van-Anh Nguyen, Dai Quoc Nguyen, Van Nguyen, Trung Le, Quan Hung Tran, and Dinh Phung. Regvd: Revisiting graph neural networks for vulnerability detection. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 178–182, 2022.

[26] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[27] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, 139:84–106, 2018.

[28] David Shriver, Sebastian Elbaum, and Matthew B Dwyer. Reducing dnn properties to enable falsification with adversarial attacks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 275–287. IEEE, 2021.

[29] Jagsir Singh and Jaswinder Singh. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Information and Software Technology*, 121:106273, 2020.

[30] Alireza Souri and Rahil Hosseini. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, 8(1):1–22, 2018.

[31] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 109–119, 2018.

[32] Sahil Suneja, Yunhui Zheng, Yufan Zhuang, Jim A Laredo, and Alessandro Morari. Probing model signal-awareness via prediction-preserving input minimization. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 945–955, 2021.

[33] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[34] Muhammad Usman, Yannic Noller, Corina S Păsăreanu, Youcheng Sun, and Divya Gopinath. Neurospf: A tool for the symbolic analysis of neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 25–28. IEEE, 2021.

[35] Wei Wang, Guozhu Meng, Haoyu Wang, Kai Chen, Weimin Ge, and Xiaohong Li. A 3 ident: A two-phased approach to identify the leading authors of android apps. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 617–628. IEEE, 2020.

[36] Yulong Wang, Tong Sun, Shenghong Li, Xin Yuan, Wei Ni, Ekram Hossain, and H Vincent Poor. Adversarial attacks and defenses in machine learning-powered networks: A contemporary survey. *arXiv preprint arXiv:2303.06302*, 2023.

[37] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. Correlations between deep neural network model coverage criteria and model quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 775–787, 2020.

[38] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):1–40, 2017.

[39] Huangzhao Zhang, Zhuo Li, Ge Li, Lei Ma, Yang Liu, and Zhi Jin. Generating adversarial examples for holding robustness of source code processing models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1169–1176, 2020.

[40] Pengcheng Zhang, Bin Ren, Hai Dong, and Qiyin Dai. Cagfuzz: coverage-guided adversarial generative fuzzing testing for image-based deep learning systems. *IEEE Transactions on Software Engineering*, 48(11):4630–4646, 2021.

[41] Quan Zhang, Yifeng Ding, Yongqiang Tian, Jianmin Guo, Min Yuan, and Yu Jiang. Advdoor: adversarial backdoor attack of deep learning system. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 127–138, 2021.

[42] Xiyue Zhang, Xiaofei Xie, Lei Ma, Xiaoning Du, Qiang Hu, Yang Liu, Jianjun Zhao, and Meng Sun. Towards characterizing adversarial defects of deep learning software from the lens of uncertainty. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 739–751, 2020.

[43] Xufan Zhang, Ziyue Yin, Yang Feng, Qingkai Shi, Jia Liu, and Zhenyu Chen. Neuralvis: visualizing and interpreting deep learning models. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1106–1109. IEEE, 2019.

[44] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. Dynamic slicing for deep neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 838–850, 2020.

[45] Zhe Zhao, Guangke Chen, Jingyi Wang, Yiwei Yang, Fu Song, and Jun Sun. Attack as defense: Characterizing adversarial examples using robustness. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 42–55, 2021.