

Early and Realistic Exploitability Prediction of Just-Disclosed Software Vulnerabilities: How Reliable Can It Be?

- ① EMANUELE IANNONE, Software Engineering (SeSa) Lab. University of Salerno, Italy
- ① GIULIA SELLITTO, Software Engineering (SeSa) Lab. University of Salerno, Italy
- ① EMANUELE IACCARINO, Software Engineering (SeSa) Lab. University of Salerno, Italy
- ① FILOMENA FERRUCCI, Software Engineering (SeSa) Lab. University of Salerno, Italy
- ① ANDREA DE LUCIA, Software Engineering (SeSa) Lab. University of Salerno, Italy
- ① FABIO PALOMBA, Software Engineering (SeSa) Lab. University of Salerno, Italy

With the rate of discovered and disclosed vulnerabilities escalating, researchers have been experimenting with machine learning to predict whether a vulnerability will be exploited. Existing solutions leverage information unavailable when a CVE is created, making them unsuitable just after the disclosure. This paper experiments with *early* exploitability prediction models driven exclusively by the initial CVE record, i.e., the original description and the linked online discussions. Leveraging NVD and Exploit Database, we evaluate 72 prediction models trained using six traditional machine learning classifiers, four feature representation schemas, and three data balancing algorithms. We also experiment with five pre-trained large language models (LLMs). The models leverage seven different corpora made by combining three data sources, i.e., CVE description, SECURITY FOCUS, and BUGTRAQ. The models are evaluated in a *realistic*, time-aware fashion by removing the training and test instances that cannot be labeled “*neutral*” with sufficient confidence. The validation reveals that CVE descriptions and SECURITY FOCUS discussions are the best data to train on. Pre-trained LLMs do not show the expected performance, requiring further pre-training in the security domain. We distill new research directions, identify possible room for improvement, and envision automated systems assisting security experts in assessing the exploitability.

CCS Concepts: • **Security and privacy** → **Software security engineering**; • **Software and its engineering** → **Software maintenance tools**.

Additional Key Words and Phrases: Exploitability Prediction, Software Vulnerabilities, Machine Learning, Text Mining, Natural Language Processing.

ACM Reference Format:

① Emanuele Iannone, ① Giulia Sellitto, ① Emanuele Iaccarino, ① Filomena Ferrucci, ① Andrea De Lucia, and ① Fabio Palomba. 2024. Early and Realistic Exploitability Prediction of Just-Disclosed Software Vulnerabilities: How Reliable Can It Be?. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2024), 41 pages. <https://doi.org/10.1145/3654443>

Authors' addresses: ① Emanuele Iannone, eiannone@unisa.it, Software Engineering (SeSa) Lab. University of Salerno, Italy; ① Giulia Sellitto, gisellitto@unisa.it, Software Engineering (SeSa) Lab. University of Salerno, Italy; ① Emanuele Iaccarino, e.iaccarino1@studenti.unisa.it, Software Engineering (SeSa) Lab. University of Salerno, Italy; ① Filomena Ferrucci, fferrucci@unisa.it, Software Engineering (SeSa) Lab. University of Salerno, Italy; ① Andrea De Lucia, adelucia@unisa.it, Software Engineering (SeSa) Lab. University of Salerno, Italy; ① Fabio Palomba, fpalomba@unisa.it, Software Engineering (SeSa) Lab. University of Salerno, Italy.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1049-331X/2024/1-ART1 \$15.00
<https://doi.org/10.1145/3654443>

1 INTRODUCTION

Software vulnerabilities represent serious threats to software dependability, allowing malicious users to attack its confidentiality, integrity, or availability [39, 69]. Vulnerabilities require specific methods and techniques to be detected [7, 62] and removed [18] promptly to avoid undesired consequences [32]. However, not all vulnerabilities are the same, as their exploitation may require different expertise and may have variable consequences. Since the number of newly discovered vulnerabilities increases rapidly [38], both vendors (i.e., owners of vulnerable products) and clients are interested in obtaining reliable feedback on the risk that a vulnerability may be exploited. The availability of such feedback is useful for various tasks, including prioritizing security verification [36, 81] and identifying suitable preventive actions to reduce the risk of an attack [30], such as the replacement of vulnerable constructs or APIs [47, 56].

Researchers have envisioned novel methods to estimate the dangerousness of newly discovered vulnerabilities. The most basic mechanisms are driven by the CVSS (Common Vulnerability Scoring System¹) “Base” score assigned. In particular, a vulnerability receiving a higher CVSS “Base” score is deemed more severe than others. To a certain extent, it is considered to have a higher chance of being exploited soon. Unfortunately, such an assumption does not always hold: a higher severity score does not necessarily imply a more likely exploitability, nor do lower scores denote less risk of being exploited [1]. For instance, this is the case of the infamous HEARTBLEED bug (CVE-2014-0160), whose CVSS 2.0 “Base” score was 5.0 (translating into medium risk), way lower than other vulnerabilities that were never observed to be exploited in the wild [32]. The upgraded version of CVSS, i.e., version 3.0, added a number of new metrics to capture additional aspects that were neglected in the previous version and corrected some inaccuracies. The new and improved “Base” score formula could address scenarios like the one observed for HEARTBLEED. Yet, this new version still cannot be used as a proxy for the exploitability risk. Indeed, CVE-2014-6049 received a CVSS 3.0 “Base” score of 2.7 (i.e., low risk), but was exploited less than a week after its disclosure. Such a score was even lower than the CVSS 2.0 “Base” score, i.e., 5.5. Therefore, it is unfortunate to note that CVSS 3.0 does not solve all the issues affecting version 2.0. Moreover, there is no significant correlation between CVSS “Base” score and the exploitability risk of a vulnerability. This alarming lack of correlation also happens for the more specific “Exploitability” and “Impact” sub-scores—i.e., the partial metrics required to compute the final “Base” score. Further detail about the correlation between the CVSS metrics and the risk of exploitability is available in the online appendix of this paper [49]. In summary, the values of the CVSS metrics assigned to the vulnerabilities affecting a system cannot provide a useful estimation of the probability of the system being attacked.

An alternative approach to gain insights on the risks associated with a vulnerability consists in adopting various strategies based on machine [14, 17, 85] and deep [45] learning models. The goal is to predict whether a new vulnerability will be exploited, either labeling it as “likely exploitable” or estimating a probability of exploitation [51] using a number of predictors from different data sources. In this respect, researchers have been using many sources of information connected to a vulnerability, ranging from its brief description contained in the CVE (Common Vulnerabilities and Exposures) record—i.e., a data structure containing all the information linked to a disclosed vulnerability—to online mentions from social media or the dark web [2, 85]. The vast majority of the proposed models rely on all the complete CVE information that were obtainable when the datasets used for the experimentations were built, hence also including the data that became available days or weeks after the official disclosure of a new vulnerability—for instance, CVE-2020-0583 received the CVSS “Base” score only 10 days after the disclosure. Indeed, as soon as a new CVE is added to the official database, it is only provided with a short description and at least one public reference,

¹CVSS website: <https://www.first.org/cvss>

as required by CVE [35]. Thus, all the CVSS scores, the appropriate CWE (Common Weakness Enumeration²) and CPE (Common Platform Enumeration³) that have been leveraged so far for exploitability prediction tasks cannot be used *before* the results of the in-depth analysis made by security experts are available. During the period ranging from the vulnerability disclosure and the expert’s analysis—which can even last about two months—the existing prediction models cannot get these data from anywhere, and therefore are inoperable in practice. Developers are hence left disoriented, without any estimate of the risk associated with the new vulnerability. However, the phase immediately following the disclosure of a new vulnerability is the most alarming, as practitioners must take countermeasures as soon as possible; therefore, even a small hint of its possible exploitability would be beneficial to dedicate the right effort to its resolution. We recognize the need for an exploitability analysis of software vulnerabilities to provide developers and security experts with preliminary information that could be used to assess the dangerousness of a newly disclosed vulnerability and immediately take appropriate preventive actions to limit the potential harm of an attack.

In this work, we aim to investigate the effectiveness of *early* exploitability prediction models that exclusively rely on what we refer to as “*early data*”, i.e., the data already available in the CVE record of a *just-disclosed* software vulnerability, to determine whether it will be exploited in the future. An *early* prediction model leverages only those pieces of information that were already published *before* the disclosure date, i.e., the short description and the referenced *online discussions*, e.g., mailing lists and security advisories, and in no way relies on further analyses performed by security experts or additional data to be retrieved from the vulnerable system. Our goal is to experiment with early exploitability prediction modeling to assess whether and to what extent the dangerousness of a vulnerability can be estimated with sufficient confidence as soon as the vulnerability is disclosed for the first time.

To achieve our goal, we first collect all known vulnerabilities and exploits in the NATIONAL VULNERABILITY DATABASE (NVD) and the EXPLOIT DATABASE (EDB), respectively, at the time we started this study, and enrich them with the data coming from the online discussions mentioning them. Such data are joined in seven combinations to build the text corpus from which the prediction models extract the textual features. Then, we experiment with a total of 72 models, made from the combination of six different machine learning (ML) algorithms, three data balancing settings, and four different ways to encode unstructured text into features, and we investigate the employment of five pre-trained Large Language Models (LLMs), to determine which is the best solution to employ for this kind of task.

In addition, we evaluate all these models in a *realistic* scenario, i.e., simulating a real software production environment to investigate whether they can be effective in practice. To do this, we validate the models pretending to build and deploy them at different points in time, i.e., *reference dates*, and we assess how their performance changes with the evolution of the software history. In particular, we apply a *time-aware* validation mechanism, sorting the full dataset of CVEs by disclosure date, and splitting it at the reference dates. At each round of validation, we use all the data before the reference date to build the dataset fold for the round—this simulates the models’ deployment scenario in which all the information available from the past is leveraged for the learning, and the base of knowledge grows over time. To evaluate the models, we split each fold into training and test sets, ensuring that the vulnerabilities in the test set are published after those falling into the training set. Then, we adopt a data labeling strategy similar to the one explained by Jimenez et al. [54], i.e., we mark the training instances as “*exploitable*” only if they were exploited

²COMMON WEAKNESS ENUMERATION: <https://cwe.mitre.org>

³COMMON PLATFORM ENUMERATION: <https://nvd.nist.gov/products/cpe>

before the train-test split date, and we mark the test instances as “*exploitable*” if they were exploited before the reference date. However, we recognize that not all the information collected from the past is always reliable. In fact, since vulnerabilities are exploited over time, the CVEs published close to the reference date that were not exploited yet cannot be confidently labeled as “*neutral*”, as no sufficient time has elapsed to let the first exploits arise. Therefore, we further clear the dataset fold from those vulnerabilities falling into such an “uncertainty window” with no associated exploit yet. This data labeling strategy—detailed in Section 3—aims to provide a more realistic evaluation of the early exploitability prediction models.

The results show that the text from online discussions, particularly from SECURITY FOCUS, can significantly boost the effectiveness of prediction models that leverage only the CVE initial description. All traditional ML models reached their best performance with vulnerability data before 2010. Oversampling the training data generally benefits all models, which draw the best performance when the features are weighted by their frequency (i.e., Term Frequency). The classifier with the best trade-off is the LOGISTIC REGRESSION, reaching a weighted F-measure of 0.49 and weighted MCC of 0.36. The most precise classifier is the RANDOM FOREST, while the one having the highest recall is KNN. The pre-trained LLMs used as-is failed to perform well, behaving like constant classifiers always predicting “*neutral*”.

Based on the results obtained, we envision a set of research directions that aim to (i) improve the quality of exploitability prediction models with particular attention to the data quality—i.e., the ground truth choice, the feature representation, etc.—and (ii) integrate exploitability prediction models into existing vulnerability assessment pipelines to better support security analysts. The experimented prediction models only assess the exploitability of publicly disclosed vulnerabilities without addressing undisclosed vulnerabilities due to inaccessibility to information that enables the prediction [13]. Our key contributions can be summarized as follows:

- i An **evaluation method** to assess the effectiveness of *early* exploitability prediction models that exclusively rely on the data available in a CVE record at the time of its public disclosure;
- ii A **data cleaning strategy** to remove those data instances that cannot be labeled with sufficient confidence since not enough time has passed since the vulnerability disclosure date.
- iii A **data collection procedure** to mine and aggregate online discussion data referenced by the external links in the CVE records, resulting in a novel dataset that researchers can use for further analyses.
- iv An **empirical comparison** of how different configurations of an ML pipeline can influence the performance of an *early* exploitability prediction model, involving 72 traditional learning algorithms, three balancing settings, and four feature representation techniques, and also five pre-trained LLMs used as-is.
- v A **comprehensive dataset**—which we publicly release [49]—containing information about disclosed vulnerabilities (CVEs), the linked initial mentions in online sources like SECURITY FOCUS and BUGTRAQ, and their public proof-of-concept exploits.
- vi A **reproducible pipeline** for training, validating, and analyzing all the ML models implemented as a collection of PYTHON scripts, made available in an **online appendix** [49].

In this paper, we focus on the feasibility of building *early* exploitability prediction models, evaluating their performance in a realistic setting. We highlight that our work is not intended to provide practitioners with a ready-to-use solution, but rather to take the first steps into the empirical investigation of *early* exploitability prediction, which can be finally put into practice with further research effort spent by the community.

Structure of the paper. Section 2 presents background information on the life cycle of software vulnerabilities, other than discussing the related literature and the limitations we aim to address.

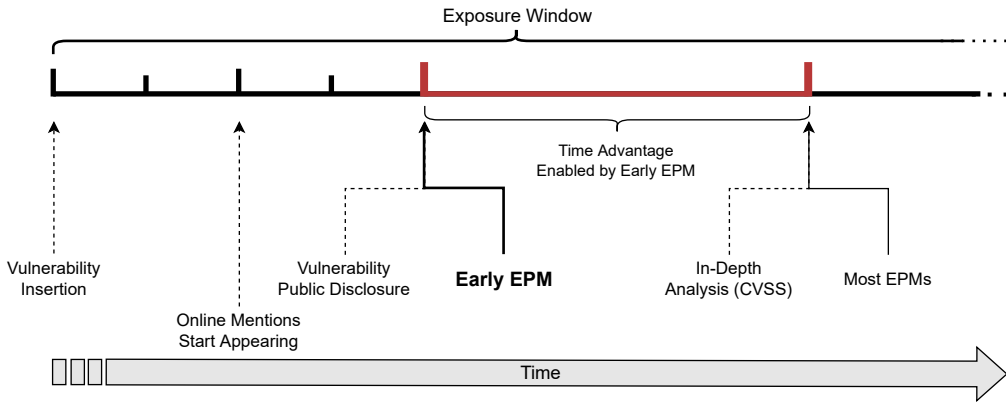


Fig. 1. The applicability moment of our exploitability prediction model (EPM) on the general vulnerability life cycle, also compared with the applicability of other solutions in literature [2, 12, 14, 34, 51, 66, 85]. The red line represents the delay between the moment a vulnerability is publicly disclosed and the CVSS analysis made by experts.

Section 3 reports the research questions driving our work and the methods we applied to address them, while Section 4 presents the results we observed. Section 5 further elaborates on the insights of the study and the implications for researchers and practitioners. The potential threats to the validity of the study are discussed in Section 6. Finally, Section 7 concludes the paper, outlining our future research agenda.

2 BACKGROUND AND RELATED WORK

2.1 Software Vulnerabilities Life Cycle

MITRE defines a software vulnerability as a flaw in a software component caused by a source code weakness that malicious users can exploit to cause damage to the confidentiality, integrity, or availability of the impacted components.⁴ According to this definition, a vulnerability is an identifiable instance of a defect affecting the source code and is characterized by its life cycle inside the flawed system. This life cycle starts with introducing the vulnerability in the system, meaning that there is a certain point in the lifetime of a software product in which the vulnerability is—involverarily or voluntarily—introduced in the code [48]. The vulnerability is potentially exploitable as soon as the flawed code is released, starting its *exposure window* [39]. After its insertion, the vulnerability can be identified in several ways: internally through manual code inspection or running static analysis tools [7, 96], or from external bug reports [19, 88] or online discussions on public and independent channels (e.g., SECURITYFOCUS).⁵ Once the vulnerability is discovered, the recommended action should be to warn a larger audience about the security issue affecting the software. To this aim, the vendor may request the allocation of a CVE record to have a unique identifier and a standardized descriptor of that specific flaw, which third-party security experts must approve. This certification step ensures that the reported vulnerability is valid and comes from a trustworthy source, which is why an external reference describing the issue is mandatory. The time ranging from the discovery to the public disclosure varies from vendor to vendor, depending on the policy they choose to adopt [5, 19, 105].

⁴CVE Glossary: <https://www.cve.org/ResourcesSupport/Glossary>

⁵SECURITYFOCUS website: <http://online.securityfocus.com/bid>

After the official disclosure by CVE, another external party may provide deeper analyses on the nature of the vulnerability, for example, by identifying the appropriate weakness type (CWE), determining the affected vulnerable configurations (CPE), and estimating its exploitability and potential impact. The famed framework CVSS—defined and managed by the Forum of Incident Response and Security Teams (FIRST)—is commonly adopted to measure this latter aspect. The CVSS standard defines three metrics groups (“*Base*”, “*Temporal*” and “*Environmental*”) that capture different aspects of a vulnerability, each containing a set of ordinal-scale metrics. The most relevant ones belong to the “*Base*” group, which is meant to stay stable after the initial assessment—in contrast with the “*Temporal*” and “*Environmental*” ones that are subject to continuous updates. In particular, the “*Base*” metrics values can also be aggregated together to form a continuous value ranging from 0 to 10, known as the “*Base*” score, that summarizes the overall severity of the vulnerability. This score is computed using a conversion table by remapping each value to a pre-defined number. The formula is slightly different between versions 2.0 and 3.0—indeed, the latter considers the user interaction and the degree of privileges required to carry out successful exploitation. At the end of October 2023, the newest 4.0 version of CVSS will be publicly released, and the process of adapting all the scores will begin. Despite its popularity, CVSS measurement has not been exempt from criticism. One of its main weak points is its subjectiveness, caused by a lack of clearly defined criteria for assigning the correct value to each metric. Indeed, the scoring procedure is driven by a set of self-asked questions about the characteristics of a vulnerability. In this respect, different analysts might interpret these criteria differently based on their knowledge and experience, causing the final base score to vary among raters. Such a scenario might seriously threaten vulnerability management since most strategies rely on the CVSS severity scores to establish the order according to which the security issues must be handled [41]. Figure 1 shows the typical life cycle of a software vulnerability, from the moment it is inserted in the code.

What is more, in a non-negligible number of cases, the CVSS measurement is not available until several days after disclosing a CVE, which can be too late in certain unfortunate cases [35]. Specifically, if we consider all the vulnerabilities published before 2021, the average delay between the disclosure and the first CVSS 2.0 measurement is 20 days. This delay becomes even greater for CVSS 3.0, reaching 69 days. In particular, in 2018, the delay was beyond the average: 38 and 96 days for CVSS 2.0 and 3.0, respectively. This phenomenon likely happens because the amount of newly disclosed vulnerabilities is continuously increasing—over 15,000 CVEs in the sole 2018—causing the experts carrying out the CVSS measurement to have an increased workload—the results of such analysis are reported in the online appendix of this paper [49]. Such a delay leaves developers weaponless for a prolonged time: they have no support in deciding the best action to react to recently-disclosed vulnerabilities.

These motivations laid the foundations of the works that employed machine learning to predict the CVSS vector—or part of it—using the initial information available at disclosure time [35, 45, 55]. Despite this, it has been widely recognized that the CVSS “*Base*” score is not a valid proxy for the exploitability risk of a vulnerability; therefore, it cannot be used as a recommendation mechanism to guide vulnerability management [1, 46].

2.2 Exploitability Prediction Modeling

Several works have considered alternative ways to assess the risk of newly-disclosed vulnerabilities in the last decade, leveraging many data types extracted from different sources. Bozorgi et al. [14] have been among the first to use machine learning to predict the exploitability of known vulnerabilities based on a large variety of metadata extracted from the Open Sourced Vulnerability Database (OSVDB) and the CVE LIST. They trained an SVM-based binary classifier to predict whether a given vulnerability is likely to be exploited in the (near) future, using a dataset of over ten thousand CVE

records disclosed between 1991 and 2007. Their model used a wide spectrum of predictors, ranging from numerical features—e.g., the CVSS metric values, the disclosure date—to the binary occurrence (presence/absence) of the tokens extracted from all the text fields using the bag-of-words (BoW) method—e.g., the CVE description, the product name, the list of external references. They labeled as “*exploitable*” (positive class) those vulnerabilities that indicated their exploitation status in their metadata, leaving the rest as “*unexploitable*” (negative class). They provided two validations: one in an offline setting—i.e., splitting the entire dataset into training and test sets randomly—and another one in an online scenario—i.e., re-training the model multiple times using only the vulnerabilities disclosed before a given date. In the former case, they observed a very high accuracy ($\approx 90\%$), while in the latter, the model stabilized around 75% in the latest rounds. Furthermore, they compared the SVM’s score (i.e., the fitted model without the decision function) with the CVSS *exploitability* score of each vulnerability in the dataset, observing that the latter is not correlated to the classifier’s scores, and highlighting the poor significance of this metric value.

This work inspired many other empirical studies that investigated the performance of different learning algorithms—e.g., LOGISTIC REGRESSION and RANDOM FOREST [12]—and data sources to build the ground truth—such as EXPLOIT DATABASE or SYMANTEC ATTACK SIGNATURES [34]—both leveraging the metadata found in the CVE records. Lyu et al. [66] experimented with more sophisticated natural language processing (NLP) techniques and a character-level Convolutional Neural Network (CNN) to predict the exploitability using only the textual description associated with each CVSS value. Although most works treated the problem as a classification task, Jacobs et al. [51] presented the *Exploitability Prediction Scoring System* (EPSS) that estimates the probability that a given disclosed vulnerability will be exploited in the following 12 months using a LOGISTIC REGRESSION model. Instead of relying on traditional feature extraction techniques (e.g., Bag-of-Words), they collected the number of occurrences of a curated list of tags extracted using the RAKE tool [84]. Moreover, they navigated the external references reported in the CVE record and scraped the text from the HTML pages to further expand the amount of data available in their dataset.

The use of textual data was further investigated in other studies, experimenting with alternative ways to extract features and assign them values. Most of these studies relied on traditional information retrieval (IR) weighting schemas, such as *word counting* or *term frequency – inverse document frequency* (TF-IDF) [2, 34, 80, 85], while others exploited NLP techniques, such as WORD2VEC models [71], to provide a compact representation of the words in a document corpus [45, 66]. Sabottke et al. [85] mined text explicitly mentioning CVEs from TWITTER to expand the pool of predictors. Besides the traditional word counting, they also computed additional metrics from the tweet statistics, such as the number of retweets and replies. Similarly, Almukaynizi et al. [2] used the mentions in the dark web assigning to each extracted token their TF-IDF weight. Apparently, only Jacobs et al. [50] contemplated the text contained in the URLs in CVE records to widen the feature set but did not provide any detail on how they mined the HTML pages. Thanks to the recent advances in Large Language Models (LLMs), Yin et al. [100] experimented with a neural network based on BERT architecture [29], which they called ExBERT (Exploit BERT) leveraging only the CVE description. To develop such a solution, they added a pooling layer and a Long-Short Term Memory (LSTM) classifier on top of a pre-trained uncased base BERT.⁶ The full model was fine-tuned on a dataset built using the information retrieved from NVD and EXPLOIT DATABASE. Each input text (i.e., the CVE descriptions) was encoded using the WORDPIECE tokenizer [98], which is suitable for text containing low-frequency words like CVE descriptions. The authors decided not to include the CVSS scores in the input text, as they are not correlated with the presence of an exploit in the EXPLOIT DATABASE. Despite obtaining encouraging results, i.e., $\sim 90\%$ F-measure, we believe that

⁶ Available at: <https://github.com/google-research/bert>

Table 1. Summary of the main works experimenting with machine learning to predict the existence of a vulnerability exploit in the future. The main novelty points are reported in **boldface**.

Study	Data Sources	Feat. Data	Feat. Representation	Early	Classifiers	Validation	Evaluation
Bozorgi et al. [14]	OSVDB, CVE LIST	CVE Metadata, CVSS, etc.	Numeric, Bag-of-Words	No	SVM	Random and Time-aware Iterative	Accuracy
Bhatt et al. [12]	NVD, EDB	CVSS, CWE, Software Type	Categorical	No	Several	Random Iterative	Accuracy
Edkrantz et al. [34]	NVD, EDB, RAPID7, SYMANTEC	CVE Metadata, CVSS, etc.	Numeric, Categorical, Bag-of-Words	No	Several	Random Iterative	Accuracy
Lyu et al. [66]	NVD, EDB, etc.	CVE Metadata	Character Embedding	No	CNN	Time-aware Non-Iterative	F-measure
Jacobs et al. [51]	NVD, EDB, RAPID7, etc.	CVE Metadata, URL Scraped	Bag-of-Words	No	LR	Time-aware Iterative	Precision, Recall
Sabotke et al. [85]	NVD, EDB, TWITTER, etc.	CVE Metadata, CVSS, Tweets	Bag-of-Words	Partial*	SVM	Random Iterative	Precision, Recall
Almukaynizi et al. [2]	NVD, EDB, ZDI	CVE Metadata, CVSS, Dark Web	Bag-of-Words, Numerical, Categorical, Binary	Partial*	Several	Time-aware Non-iterative	Precision, Recall, F-measure
Yin et al. [100]	NVD, EDB	CVE Description	BERT	No	LSTM	Random Non-iterative	Accuracy, Precision, Recall, F-measure
Yin et al. [101]	NVD, EDB	CVE Description, CVSS	BERT, Categorical	No	Several	Time-aware Iterative	Accuracy, Precision, Recall, F-measure
Suciu et al. [93]	NVD, BUGTRAQ, TWITTER, etc.	PoCs, Tweets, etc.	AST, Textual Unigrams	Incremental**	NN	Time-aware Iterative	Precision, Recall
This work	NVD, EDB, BUGTRAQ, SECURITY FOCUS	CVE Metadata, URL Scraped	Bag-of-Words, BERT, Word Embedding	Full	Several	Time-aware Iterative	F-measure, Precision, Recall, MCC

*"Feat." = Feature, "SVM" = Support Vector Machine, "LR" = Logistic Regression,

"CNN" = Convolutional Neural Network, "NN" = Feed-forward Neural Network

*Data between the disclosure and the first exploitation was not discarded.

**Data published after the disclosure was used when available.

the experimental setup suffers from two main issues. First, the set of non-exploitable instances (i.e., the vulnerabilities having no known exploit in the EXPLOIT DATABASE) were down-sampled to match the number of exploitable instances, reaching an unrealistic balanced dataset, which also affected the test set composition. Second, the WORDPIECE tokenizer was fitted on the entire dataset before splitting it into train and test subsets, influencing the vocabulary with information that was supposed not to be seen at that stage. In the end, the final scores were inflated, hindering the realism of the model's real performance.

Following a different strategy, Sabotke et al. [85] and Almukaynizi et al. [2] considered selecting part of the information published before the date on which a CVE was exploited in the wild—following the recommendation described by Reinthal et al. [80], who claimed that any realistic exploitability prediction model should not leverage the data arriving after the exploitation. The use of “future” data to predict data belonging to the “past” is just one of the limitations that Bullough et al. [17] identified in many works on the matter; in this respect, the authors presented challenges for a realistic machine learning-based exploitability prediction model. Previous studies have legitimately considered the effect of the imbalance ratio, but the re-sampling algorithms were inappropriately applied to the entire dataset, affecting the test sets as well [74]. Similar to the limitations found by Reinthal et al. [80], many works used all the data found in the CVE records when they were mined, including all the subsequent updates they received even after the exploitation. This approach is not representative of what would happen in reality. Indeed, a realistic prediction should be executable as soon as a new CVE is published, without waiting for the availability of additional metadata like the CWE, the CVSS score, or the CPE, which are known to be added only some days or weeks after the disclosure, as observed by Elbaz et al. [35]. To clarify such a concept, Figure 1 depicts the moment from which a realistic exploitability prediction model should be employed when related to the typical life cycle of a vulnerability.

2.3 Realistic Validation of Machine Learning Models for Security

In the context of vulnerability prediction modeling, Jimenez et al. [54] investigated the realism of machine learning models when dealing with data having some form of temporal relationships. The

authors shed light upon what they called the “*perfect labeling assumption*” that most researchers adopt when training a vulnerability prediction model. According to this assumption, researchers implicitly suppose that all kinds of information about any given vulnerability are always available at any time. Under this assumption, all the models would exhibit optimistic results that will likely contradict the models’ performance when they are put into production. To overcome this bias, Jimenez et al. [54] proposed an alternative approach to evaluate a machine learning model’s performance in a more realistic and reliable way. Firstly, whenever the data exhibits some form of time relationship, the authors recommend the employment of a *time-aware* validation, which takes into account the moment in which each piece of information becomes available. The models should be trained on “past” data and tested on subsequent data, i.e., avoiding the use of any fully-random validation strategies—such a recommendation is in line with those identified by Bullough et al. [17]. Then, the labels assigned to the data appearing in the training set should strictly reflect what is known at *training time*, i.e., the date used to split the training and test sets, without leveraging any future information. This approach is what they called “*real-world labeling*”, which adds more realism to the models’ validation, making their results more faithful to those scored during the production. Furthermore, they observed that instances in the negative class should be labeled as “*neutral*” rather than “*not vulnerable*”, as the problem of vulnerability prediction is undecidable [58, 82].

Jimenez et al.’s considerations [54] were embraced by several subsequent works on vulnerability prediction. Liu et al. [63] introduced the *n-fold time-series* validation, in which the dataset is sorted by date and split in $n+1$ folds to perform n evaluation rounds. At each round i , the folds from 1 to i are used as the training set, while the fold $i+1$ as the test set. A similar approach was followed by Le et al. [60], who also considered an additional fold for the validation phase. Pornprasit and Tantithamthavorn [75] considered the time constraints of vulnerability data availability. They performed a *time-aware* validation by splitting the sorted dataset into two subsets: 80% of the entries were used for the training phase and the remainder 20% for the testing phase. All the above approaches led to observing a lower performance of the models compared to those reported in time-agnostic experimental settings, as Rakha et al. [78] argued in previous work.

In the context of exploitability prediction, Yin et al. [101] tackled the “*concept drift*” problem that can negatively influence the models’ credibility. Indeed, information about the existence of exploits for vulnerabilities changes over time, i.e., a vulnerability can be exploited years after its disclosure, with an inevitable impact on the labels assigned to each instance. Hence, they proposed an incremental learning strategy called Real-time Dynamic Concept Adaptive Learning (RDCAL) that trains and evaluates the models in an online learning scenario (i.e., when the models are re-trained multiple times on time-ordered data) where the labels (i.e., the exploitability status) are determined at each iteration rather than upstream. They experimented with this strategy using four classifiers, achieving comparable performance with the traditional batch learning schema. Similarly, Suciu et al. [93] observed that additional vulnerability data becomes available over time, e.g., write-ups, PoCs, and social media discussions, which often provide meaningful information about the likelihood of exploits happening in the future. Driven by the intuition that the probability of observing an exploit changes over time, they proposed a random variable called *Expected Exploitability*. Instead of deterministically labeling a vulnerability as “*exploitable*” or “*not exploitable*”, they considered exploitability as a stochastic process, describing the likelihood over time that a functional exploit will be developed for the considered vulnerability. Such *Expected Exploitability* metric can be computed leveraging supervised machine learning based on historical patterns observable for similar vulnerabilities, and be updated continuously by adding new information to the model as soon as it becomes available, e.g., CVSS metrics can be used for updating the prediction as soon as they are published. They evaluated the precision and recall of their classifier against five baseline

models, observing that *Expected Exploitability* outperforms existing metrics for exploitability prediction, and improves over time.

2.4 Our Contribution

We believe all the considerations made for realistic vulnerability prediction modeling hold for exploitability prediction as well—for instance, we cannot look at the exploits that appeared after the training time to label a vulnerability as “*exploitable*” or “*neutral*” in the training set. In this work, we evaluate several *early* exploitability prediction models that leverage only the description and the online discussion data available at the *disclosure time* of a new vulnerability. We evaluate the models using a *time-aware* realistic validation schema: we make several training and testing rounds where we (i) clear out the instances for which the labeling cannot be done with sufficient confidence, (ii) assign the labels (“*exploitable*” and “*neutral*”) with an eye to the recommendations by Jimenez et al. [54], (iii) prepare the text corpora from different sources, (iv) extract the features according to the vocabulary fitted on the training data, (v) adjust the training set using data balancing algorithms, (vi) train and test the models.

Compared to previous work on the matter, our novelty points include: (i) the creation of a full *early* exploitability prediction pipeline that only leverages the information available at the vulnerability disclosure time; (ii) the use of online discussion from SECURITY FOCUS linked via references reported in the CVE records; (iii) the analysis of the model performance with the MCC metric, which considers all four quadrants of the confusion matrix; (iv) the experimentation with word embedding to represent textual features; (v) the employment of pre-trained LLMs in the task of exploitability prediction.

Table 1 reports a summary of the main works on exploitability prediction and highlights the core differences between the existing literature and our work.

3 EMPIRICAL STUDY DESIGN

In Section 3.1, we formulate the goal of our study according to the Goal-Question-Metric (GQM) template [95]; from this goal, we distilled our research questions (RQs). In Section 3.2, we describe the steps we followed to collect the data needed to fuel the prediction models. In Section 3.3, we report the details of the models that were selected to participate in our experiments, and in Section 3.4 we explain how we trained and evaluated them. Lastly, in Section 3.5 we focus on the implementation details regarding the models, and we describe the infrastructure we employed to run our experiments.

3.1 Study Goal and Research Questions

The *goal* of this empirical study was to investigate the performance of machine learning (ML)-based classifiers to predict the exploitability of a just-disclosed vulnerability, with the *purpose* of providing early feedback on the exploitability of new vulnerabilities in a realistic scenario. The *perspective* was of both practitioners and researchers. The former are interested in obtaining as much information as possible to (i) have an initial assessment supporting the CVSS measurement and (ii) understand when and how the vulnerabilities afflicting their applications must be handled. The latter are interested in (i) comprehending the predictive capabilities of textual data—obtained from online discussions about the vulnerabilities written in natural language—represented in different ways and (ii) assessing the effectiveness of different learning configurations.

To the best of our knowledge, the current research in exploitability prediction has always involved all the data available in CVE databases at the time of the study when building the dataset used by the experimented classifiers. This scenario caused the models to rely on subjective measures like the CVSS scores to make their predictions. However, in a real-world scenario, such information is

only made available at a non-negligible distance from the CVE disclosure date [35]—on average, 20 days after the disclosure by the year 2021. We hypothesize that a realistic prediction model should only leverage *early data*, i.e., those pieces of information available at the *disclosure time*, as soon as a vulnerability is made public through a CVE record.

Therefore, our empirical investigation focused on employing ML algorithms to train classification models that recognize potentially exploitable vulnerabilities using exclusively early data. We observed that whenever a new vulnerability is discovered, the ordinary mechanism to raise awareness about it consists of opening a free commentary about it on public mailing lists or other similar discussion channels [5, 19]. Such *data sources* contain valuable information that may provide additional insight into the seriousness and impact of the vulnerability, e.g., a crash stack trace [90], the reproduction steps [20], or even code snippets showing that an exploit is feasible in principle (a.k.a. Proof of Concepts, PoCs) [92]. Whilst involving data from multiple sources could provide a more comprehensive perspective of the problem [103, 106], the effect on the accuracy of the prediction models is not always positive [67]. Indeed, the information in each source might have evident contradicting information [25] that would not allow the models to distinguish between “exploitable” and “neutral” vulnerabilities. Hence, we are interested in investigating how different combinations of data from multiple sources affect the models’ performance. We asked:

Q RQ₁. *What is the impact of the **early data source combination** on the performance of ML-based exploitability prediction?*

Extracting relevant information from vulnerability data sources that allow the models to recognize exploitable vulnerabilities is not straightforward. Not only do such sources predominantly contain *unstructured text*, but also, no automated mining tool that extracts the relevant pieces of information, e.g., the code snippets isolated accurately, exists. The only available solutions can only work with traditional bug reports and for specific programming languages, such as INFOZILLA [10] that only works for bug reports in JAVA. We must rely on text processing techniques that automatically determine the features from a corpus of natural language text [11] to let the prediction models learn from unstructured text.

There are many ways to achieve such a task, that we group into two main categories: (1) those encoding the tokens found in the corpus—after adequate pre-processing—as individual features, and (2) those that learn how to represent a given text as an *embedding*. In the first case, once the textual features, e.g., tokens or words, are extracted, they are weighted according to different mechanisms, such as by counting the number of times a certain feature appears in a document in the corpus, or by measuring the frequency of that token over the entire corpus [8]. The number of resulting features cannot be controlled directly and heavily depends on the actual content of the documents in the corpus and on the pre-processing steps taken before, e.g., *stemming*. In the second case, the features do not represent specific textual elements, but they are “latent variables” that the model infers from the input corpus [59, 71]. Unlike the first category of techniques, the size of the embeddings is generally decided upstream before launching the embedding algorithm. The choice of the specific text representation technique can greatly impact the models’ performance [28, 43].

Furthermore, we believe other elements can influence the models’ performance that are worth investigating, such as the choice of the specific learning algorithm [94] or the use of data balancing algorithms to deal with the imbalance between “exploitable” and “neutral” instances [74]. Thus, we asked:

Q RQ₂. *What is the performance of ML-based early exploitability prediction under different **learning configurations**?*

Due to the recent advancements in the field of Natural Language Processing (NLP) and the popularity of pre-trained Large Language Models (LLMs), we also wanted to assess their suitability for such a task, as experimented by Yin et al. [100]. Such models come with pre-trained weights learned in a self-supervised manner from large corpora of text not directly related to the specific tasks, e.g., general English text and/or examples of code written in different programming languages. The advantage of such models stands in their ability to determine the representation for the input (depending on what was seen during the pre-training stage) and return the prediction in a single shot. Therefore, we formulated two sub-questions to answer RQ_2 , the first one investigating the performance of several ML models made of the traditional key elements—i.e., feature representation, training data balancing, and learning algorithm—while the latter focused on the use of end-to-end pre-trained LLMs.

Q $RQ_{2.1}$. *What is the performance of ML-based early exploitability prediction under different learning configurations using **traditional ML**?*

Q $RQ_{2.2}$. *What is the performance of ML-based early exploitability prediction under different learning configurations using **end-to-end pre-trained LLMs**?*

3.2 Data Collection

The *context* of this empirical study was made of publicly disclosed vulnerabilities accompanied by references to online discussions mentioning them, such as public mailing lists and security advisories. Our literature review found no readily available dataset with all the data we need, i.e., the text of online discussion linked to disclosed vulnerabilities and the dates when the vulnerability was disclosed and exploited. Therefore, we adopted a systematic data collection procedure to link all the existing known vulnerabilities to public websites where they had been likely discussed for the first time before the official disclosure date—in the rest of the paper, we also use the wording “publication date” when referring to the date on which a CVE record is made accessible. Then, we mined a large set of public scripts and PoCs available online and linked them to the vulnerabilities they exploited to carry out (pseudo-)realistic attacks. Figure 2 depicts all the steps (from 1 to 7) we took to collect the data needed to answer our research questions, each detailed in the following. Table 2 summarizes the collected data, reporting how much information we could retrieve from each considered data source. The scripts to run the entire data collection procedure are available in the online appendix of this paper [49].

3.2.1 Mining Known Vulnerability Data. We relied on the National Vulnerability Database (NVD),⁷ a comprehensive catalog of disclosed vulnerabilities reported in the form of CVE records. NVD enriches the upstream CVE LIST managed by MITRE⁸ by adding the CVSS vectors, the labeling of known affect software versions via CPE, etc. For such reasons, NVD has been the basis of many empirical studies on software vulnerabilities, being considered a reliable source of high-quality information [48, 54, 65, 72]. Our study did not target any specific platform or programming language, so we collected all existing vulnerabilities available in NVD at the time of the study. Thus, we downloaded the full dump of NVD curated by the CVE SEARCH project⁹ on November 03, 2021, gathering 148,299 CVE entries (Step 1 in Figure 2). We pre-processed this raw dataset to ensure that the data quality was suitable for our study. In particular, we filtered out those entries that were (i) malformed (e.g., the identifier did not point to a really-existing CVE record), (ii) rejected (i.e.,

⁷NVD website: <https://nvd.nist.gov>

⁸CVE LIST website: <https://cve.mitre.org>

⁹CVE SEARCH dump: <https://www.cve-search.org/dataset>

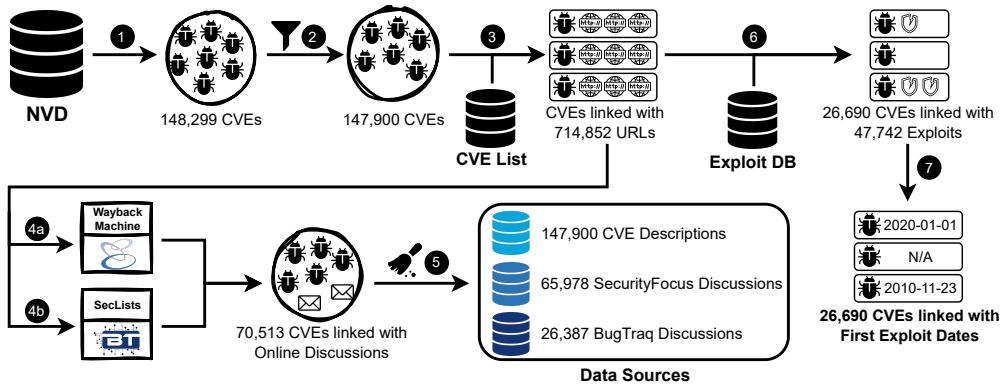


Fig. 2. Overview of the data collection process employed. The process focuses on how we collected the data from the various sources, i.e., NVD, CVE LIST, EXPLOIT DATABASE, SECURITY FOCUS, and BUGTRAQ.

the CVE identifier was allocated but never approved at the final stage), or (iii) lacking external references. We could recognize vulnerabilities falling in those cases by inspecting the content of the dump: malformed CVE identifiers did not adhere to the pattern `CVE-XXXX-YYYY`; rejected CVEs had a clear statement of their rejection in the description; CVEs lacking external references were missing a list of links in the HTML page on CVE LIST. In the end, the three filters led to the removal of just 214, 135, and five entries, respectively (Step 2 in Figure 2), resulting in 147,900 valid CVEs.

Since we were interested in using only the information available at the disclosure time, we retraced the change history of the CVE record stored in NVD; for each CVE we scraped the content of its descriptive HTML page in NVD, as it contains a set of tables reporting the changes made to the record and their dates. In doing so, we could fetch the original description that the CVE had at the time it was first disclosed, so that we could use it within our *early* exploitability prediction models in a realistic scenario—indeed, the models should not be allowed to use information that was made available years after the disclosure. It is worth pointing out that the original date on which a CVE record was created is not reported in NVD but rather on the CVE LIST website. Hence, we mined the HTML pages in the CVE LIST website as well. The scraping of HTML pages of both NVD and CVE LIST was supported by the BEAUTIFULSOUP library for PYTHON.¹⁰

3.2.2 Mining Online Discussion Data. Any CVE record is equipped with a continuously updated list of external reference URLs pointing to web pages concerning that specific vulnerability, e.g., online discussions, official patches, bug reports. In this respect, our goal was to define an exploitability prediction model that leverages only those references available at the disclosure time, as we believe they could be the reason behind the allocation request of the CVE identifier. However, the references in the NVD dump are not provided with the date they were added to the CVE records, preventing us from knowing whether they have already been linked at the time of the record allocation or at a later stage. We observed that the CVE LIST website maintains the references in a different way: each link is labeled with a special keyword indicating its type—e.g., vendor advisory, mailing list—and origin, i.e., the website it points to.¹¹ Hence, we analyzed 714,854 CVE LIST references among all the 147,900 CVE records (Step 3 in Figure 2). We observed that the most recurring keyword was *BID* (SECURITYFOCUS Bugtraq ID), which refers to security advisories published on the SECURITYFOCUS

¹⁰BEAUTIFULSOUP website: <https://beautiful-soup-4.readthedocs.io>

¹¹CVE References: <https://cve.mitre.org/data/refs/index.html>

website (43% of CVEs had at least one reference to this category). Such a website has long been considered a reliable source to report security bugs—each uniquely identified with a *BID*—and tracks existing solutions and working exploits [38]. Moreover, its plain HTML structure allowed the easy recovery of all the data using a simple HTML parser with the application of filters to exclude those references published after the vulnerability disclosure date. For all these reasons, we ignored the references listed in the NVD dump, navigated the *BID*-labeled URLs in CVE LIST, and parsed the content of the HTML in the response using BEAUTIFULSOUP—this was the only available option to recover such data, as SECURITYFOCUS does not expose any accessible API. It is worth pointing out that since February 2020, SECURITYFOCUS has stopped publishing further *BID* advisories; hence the URLs stored in the CVE records are actually inaccessible. We circumvented this limitation by exploiting the WAYBACK MACHINE¹² service provided by the INTERNET ARCHIVE library [3], which offers free access to many digital resources that were once available on the web. Therefore, we queried the API exposed by the service that returns an active URL—stored in its archives—having the same HTML content as the input URL (Step 4a in Figure 2).

We also observed that *BID* reports are commonly related to a discussion on a public mailing list known as BUGTRAQ, one of the most popular discussion boards where participants have been conducting discussions on newly-discovered vulnerabilities since 1993. All the discussions are held in natural language (commonly English) without following any specific text structure. Among the discussions, the only consistency is the header containing the original publication date. Consequently, we considered *BugTraq* references in addition to those labeled with *BID*. BUGTRAQ has encountered a similar fate to SECURITYFOCUS, as it was shut down in 2020; yet, we still considered it alongside SECURITYFOCUS as it was referenced by a non-negligible number of the CVEs we selected (14% CVEs had at least one *BugTraq* reference). All the discussions held in the past are now archived by third-party websites, such as SECLISTS,¹³ from which we downloaded all the discussions pointed by the CVE records in our context. To recover the missing mailing list discussions, we exploited the format of the *BugTraq* identifiers, made of eight digits representing the publication day according to the format YYYYMMDD, plus a short text summarizing the content of the discussion. Both the year and the month allowed us to reach a page on SECLISTS containing the list of *BugTraqs* of that period, from which we retrieved the entry that had the highest similarity—using the Gestalt Pattern Matching [79]—with the short text in the *BugTraq* identifier. Then, we mined the content of the matched discussions leveraging BEAUTIFULSOUP (Step 4b in Figure 2). To summarize, 70,513 out of 147,900 CVEs had at least one *BID* or *BugTraq* type reference, linked to a total of 65,978 *BID* references and 26,387 *BugTraq* references. We considered these numbers sufficiently high to address the research goals of our investigation.

Afterward, we made sure to discard the text of those *BID* and *BugTraq* references (i) published after the CVE disclosure date or (ii) whose format did not allow to reliably recover their publication date. This happened for 10,973 out of 65,978 *BIDs* and for 5,368 out of 26,387 *BugTraqs*. Although these two filters caused some vulnerabilities not to have any *BID* or *BugTraq* reference, we still did not discard them entirely, as the sole original description provided in the CVE record might contain enough information to predict their exploitability, as observed in similar works [45, 66].

As a result of the process of retrieving input data for our investigation, we finally considered three data sources, namely (i) CVE records, retrieved from CVE LIST, (ii) SECURITYFOCUS *Bugtraq ID* reports, restored from the WAYBACK MACHINE, and (iii) discussions on the BUGTRAQ mailing list, collected from SECLISTS website. Each of these sources provided us with textual data that we combined to perform our experiments, as explained in Section 3.3.

¹²WAYBACK MACHINE API: https://archive.org/help/wayback_api.php

¹³SECLISTS website: <https://seclists.org>

Each text underwent a set of pre-processing steps to remove irrelevant pieces of information and facilitate encoding textual features (Step 5 in Figure 2). In particular, we first employed a set of regular expressions to detect and remove data that could negatively affect the process, such as websites, URLs, e-mail addresses, PGP signatures and messages, hex numbers, and words containing repetitions of the same letters for at least three times in a row [11, 45, 61]. Second, we applied the lowercase reduction, removed non-alphabetic characters (punctuation and Unicode symbols), and split the remaining content into tokens using the whitespace as a separator. Then, we removed any English stop word [26, 91], applied the suffix stripping using the Porter’s stemmer [76], and removed those terms having less than three characters.

3.2.3 Mining Exploit Data. The CVE references labeled as *EXPLOIT-DB* in the CVE LIST point to EXPLOIT DATABASE (EDB in short),¹⁴ which is the most comprehensive collection of public exploits and Proofs of Concepts (PoCs) that explicitly target known vulnerabilities. Navigating these references could establish the links between the vulnerabilities and their exploits. However, we observed that only a minimal set of CVE records had an explicit link to EDB (i.e., 7.9%), likely owing to an improper curation of the CVE records—and not necessarily to a real lack of an exploit. Hence, similarly to Bhatt et al. [12], we rebuilt the links crossing the opposite way, connecting all the exploits in EDB to the affected CVEs using the metadata contained in the exploit entries. To this aim, we downloaded the complete list of exploits at the date of November 03, 2021, from the official GITHUB repository¹⁵ to obtain the list of valid exploit identifiers, queried the EDB website, and parsed the HTML pages—still using the BEAUTIFULSOUP library—of each exploit to collect the target CVEs, if made explicit. Note that a single exploit or PoC may target more than one, often related, vulnerability; similarly, more than one exploit may affect a single vulnerability. In the end, a total of 47,742 exploits were collected, linked to 23,690 different CVEs, corresponding to 16.02% of the total CVEs with valid data (Step 6 in Figure 2). After connecting each CVE to their exploits, we could obtain the dates on which the first exploit of the vulnerability described in the CVE was uploaded in the EXPLOIT DATABASE. We collected all these dates and considered them as the “*exploitation dates*” (Step 7 in Figure 2), which will be needed when building our ground truth (see Section 3.4).

3.3 Model Selection

Once we had collected all the required data, which is summarized in Table 2, we could select the prediction models subject to our experiments.

To answer **RQ₁**, we first considered the textual data from the three sources selected—i.e., CVE, SECURITYFOCUS (SF), and BUGTRAQ (BT)—individually to assess which one was the most helpful in predicting the exploitability of the vulnerabilities. Then we combined them by means of *string concatenation* to understand whether the data coming from multiple sources can let the models have additional information on the vulnerabilities and improve their accuracy. Hence, we formed four combinations, i.e., ⟨CVE + SF⟩, ⟨CVE + BT⟩, ⟨SF + BT⟩, ⟨CVE + SF + BT⟩. In the end, we tested with a total of **seven** different combinations of data sources, which we call *corpora* from now on.

Then, regarding **RQ_{2.1}**, we experimented with 72 traditional learning configurations, determined by the combination of:

- **Six** machine learning algorithms, opting for the most adopted for training binary classifiers, i.e., (i) LOGISTIC REGRESSION (LR) [70], (ii) NAÏVE BAYES (NB) [83], K-NEAREST NEIGHBORS (KNN) [104], (iv) SUPPORT VECTOR MACHINE (SVM) [24], (v) DECISION TREE (DT) [16], and (vi) RANDOM FOREST (RF) [15].

¹⁴EDB website: <https://www.exploit-db.com>

¹⁵EDB repository: <https://github.com/offensive-security/exploitdb>

Table 2. Summary of the data collected from the considered data sources.

Description	Count	%
CVEs from NVD	148,299	–
Malformed	214	0.14%
Rejected	135	0.09%
No external references	5	0.003%
w/ Valid Data	147,900	99.73%
w/ Discussion	70,513	47.55%
↔ on SECURITYFOCUS	64,120	43.24%
↔ on BUGTRAQ	21,540	14.52%
↔ on both SECURITYFOCUS and BUGTRAQ	15,147	10.21%
w/ PoC in EXPLOIT DATABASE	23,690	15.97%
URLs referenced by CVEs	714,854	–
SECURITYFOCUS Discussions	65,978	9.23%
↔ Discarded	10,973	1.53%
↔ Valid	55,005	7.69%
BUGTRAQ Discussions	26,387	3.69%
↔ Discarded	5,368	0.75%
↔ Valid	21,019	2.94%

- **Four** feature representation schemas, three of which encode each token found in training set as an independent feature—i.e., the simple *word counting* (a.k.a. Bag-of-Words, BoW), *term frequency* (TF), *term frequency-inverse document frequency* (TF-IDF)—and one that automatically learns embeddings from the training set—i.e., *DOC2VEC* (DE).
- **Three** ways for managing the data imbalance during the training stage, i.e., leaving the data untouched (*Original*), over-sampling with SMOTE [21], and under-sampling with NEARMISS (version 3) [102].

Similarly, for **RQ_{2.2}**, we involved **five** pre-trained LLMs, all based on the BERT architecture [29]. Specifically, we selected: (i) DISTILBERT [87], (ii) ALBERT [57], (iii) XLM-ROBERTA [23], (iv) CODEBERT [37], and (v) CODEBERTA [64]. We selected such models because of their noticeably different pre-training backgrounds. Indeed, all models we selected have a general understanding of the English language, which was required as the text of CVE descriptions and the other data sources we considered, i.e., SECURITYFOCUS and BUGTRAQ, were also in English as well. Both ALBERT and DistilBERT were pre-trained on BOOKCORPUS¹⁶ and ENGLISH WIKIPEDIA¹⁷ corpora (just like the vanilla BERT), while XML-ROBERTA was pre-trained on COMMONCRAWL¹⁸ corpus containing text from 100 languages. Yet, vulnerability data often include elements that are not part of a common text in the English language, such as code snippets and many punctuation characters; for this reason, we also included two models having experience with programming languages, i.e., CODEBERT and CODEBERTA. Such models were pre-trained on CODESEARCHNET¹⁹ corpus, containing examples of methods from six programming languages, as well as their associated documentation (e.g., JAVADOC), generally written in English. To allow the models to be fine-tuned on our binary classification downstream task, we equipped them with a linear layer on top of the pooled output.

¹⁶BOOKCORPUS corpus: <https://yknzhu.wixsite.com/mbweb>

¹⁷ENGLISH WIKIPEDIA corpus: <https://www.english-corpora.org/wiki/>

¹⁸COMMONCRAWL corpus: <https://commoncrawl.org/>

¹⁹CODESEARCHNET repository: <https://github.com/github/CodeSearchNet>

To better contextualize the models' performance, we also involve **four** baseline models, meant to determine the real usefulness of the non-trivial models. We selected: a *Random* (RND) classifier, stating that a vulnerability is "exploitable" with 50% probability, a *Pessimistic* (PES) classifier, always predicting that a vulnerability is "exploitable", an *Optimistic* (OPT) classifier, always predicting that a vulnerability is "neutral", and a *Stratified* (STR) classifier, predicting the exploitability with a probability equal to the frequency of "exploitable" instances in the training set. Due to their nature, the baseline models ignore any feature representation and data balancing technique employed.

To summarize, we experimented with a total of 567 models, i.e., 72 traditional ML models, five LLMs, and four baseline classifiers, all trained and tested on seven corpora.

3.4 Model Evaluation Framework

To ensure high realism in our experimentation, each of the 567 models was validated in the context of a *time-aware validation*, which emulates a scenario where the prediction models are iteratively re-trained and validated at different *reference dates* with different data. Such reference dates represent the moment in which the models would be put into production. To this end, we had first to sort all the instances (i.e., the vulnerabilities) previously collected (Section 3.2) by their *publication date*, then create the folds to form the validations rounds, and finally determine the target *labels* (i.e., the expected values the models should predict) to assign to each instance according to their *exploitation date*, if any. Given the time-aware nature of the validation, the assignment of the labels could not be done for all the instances in a single shot, as it would break its realism, since exploitability data is not always available. To better clarify this concept, let us suppose we wanted to validate the models in December 2005: we would not be allowed to look into any piece of information that came out after this date. For example, if a CVE had been published in 2003 and was first seen exploited only in 2006, we must treat that instance as "*neutral*" in December 2005. Therefore, we assigned the labels to the instances at each round of validation, in line with the recommendation by Jimenez et al. [54].

We also took into account other noise-introducing factors that could affect the labeling activity, which we explain in more detail in Section 3.4.1. The way we split the entire collection of vulnerabilities into folds to create the rounds for the time-aware validation is explained in Section 3.4.2, while in Section 3.4.3 we report how we built the training and test sets for each round. Lastly, in Section 3.4.5, we describe how the models' performance was assessed. Figure 3 summarizes the entire framework with which we trained and tested the 567 models.

3.4.1 Data Cleaning Strategy. Each instance in the dataset had to be labeled according to the presence of an exploit in EXPLOIT DATABASE. The most straightforward strategy would have been to mark as "*exploitable*" (true class) those instances having at least one associated exploit and as "*neutral*" (false class) all those not having any reported exploit at all. This would have caused 16.02% vulnerabilities to be labeled as "*exploitable*" and 83.98% as "*neutral*". However, such an approach is improper in the context of a realistic validation as it would produce a large number of instances with inappropriate labels. Let us consider the case of CVE-2020-14340, published in June 2021. By the time of this part of the study, i.e., the reference date is November 2021, only five months had passed since its publication, so it was quite expected that an exploit was not already present in EXPLOIT DATABASE, as not enough time had passed since its publication to observe the first public exploit. As a matter of fact, the average time between the disclosure and the first exploitation of a vulnerability is 194 days, i.e., more than half a year. Marking as "*neutral*" such a recently-disclosed vulnerability would be *too eager*, causing an overabundance of false labels. We hypothesize that if we concede some more time to make the exploits emerge, we could label the instances more confidently. In other words, if recently disclosed vulnerabilities have not been seen

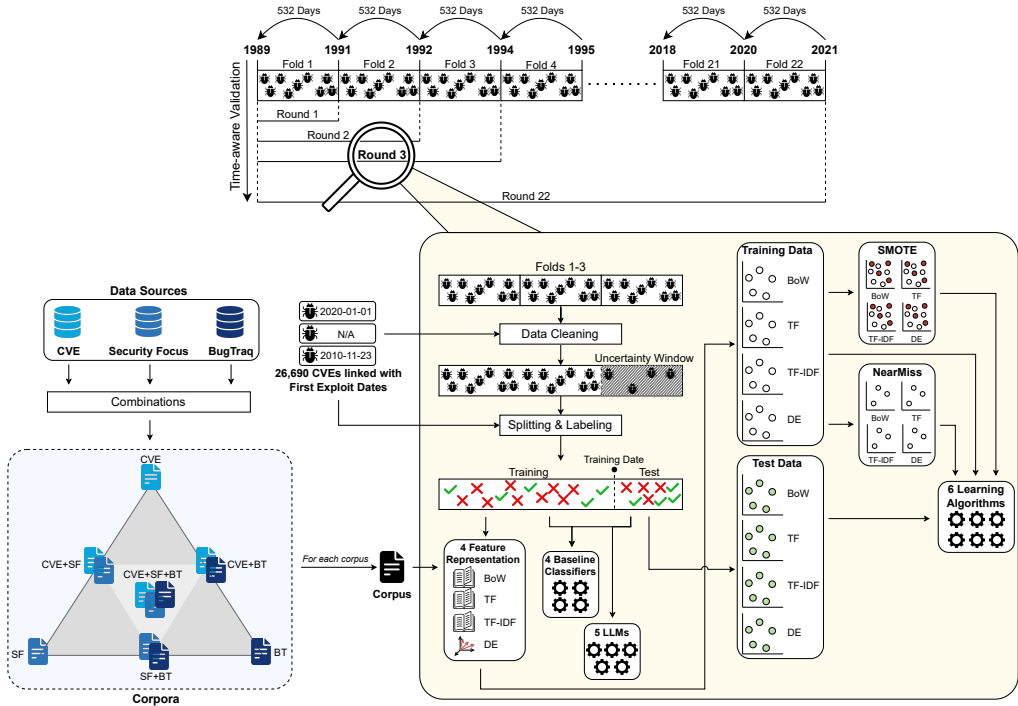


Fig. 3. Overview of the framework employed to train and test the 567 early exploitability prediction models in a realistic scenario.

exploited yet, we cannot deem them as “exploitable” or “neutral” with sufficient confidence. Thus, we decided to completely remove those instances from the validation round having the reference date of November 2021 to avoid introducing data with noisy labels—following an analogous strategy adopted by Garg et al. [42]. It is worth pointing out that such instances should not be used either as training data or as test data because, in the first case, they would inflate the number of false instances, while in the latter case, they would distort the models’ real performance.

We observed that the number of instances that risk being labeled improperly strongly depends on the amount of time we are willing to “concede” for exploits to emerge. Let us consider the case of CVE-2020-25649, which had been published six months before CVE-2020-14340 (seen in the previous example). Similarly to the previous case, half a year was not enough to let its first exploit manifest—indeed, this is even below the average exploitation time of 194 days. To minimize the risk of having improperly labeled instances among our train and test data, we selected the 90th percentile from the exploitation time distribution—corresponding to 532 days (about one year and a half)—to be the “tolerance period” we concede to exploits to manifest. Specifically, given the reference date D_i in a validation round R_i , we applied our cleaning strategy to all vulnerabilities that have been published within 532 days from D_i . All vulnerabilities within this *uncertainty window* that were not exploited before D_i were completely excluded from the round R_i . All the vulnerabilities that passed the cleaning step and those outside the uncertainty window were labeled according to the presence of an exploit in the EXPLOIT DATABASE reported before D_i .

3.4.2 Validation Round Creation. To determine the number of folds into which the dataset must be divided, and so forming the validation rounds, we used the duration of the uncertainty window set before (Section 3.4.1). Namely, starting from November 2021 (the time of this part of the study) we repeatedly went “back in time” by 532 days at a time until the date in which the first vulnerability in the dataset was published (i.e., 1989). In this way, we ended up with 22 folds, each made of vulnerabilities published in 532 days time span. For instance, the 22nd split consists of all the CVEs published between May 19, 2020, and November 2, 2021, while the 21st split consists of the CVEs published between May 18, 2020, and December 4, 2018, and so forth. Such a splitting allowed us to evaluate the models’ behavior when the uncertainty window is made of wholly different sets of vulnerabilities. Figure 3 shows an example of what happens within each round of the time-aware validation. We observe that the 22nd round, i.e., the last one, corresponds to the case in which we use the entire dataset of vulnerabilities mined in this work to train and test the models.

3.4.3 Training & Test Set Preparation. At each validation round R_i , we used all folds from 1 to i to form the dataset of round R_i . Then, we apply the data cleaning strategy described in Section 3.4.3 for all the vulnerabilities admitted in round R_i , and we created the training and test sets using a *time-aware* 80/20 splitting, i.e., placing the first 80% instances in the training set and the remaining 20% in the test set, ensuring that all the training instances were published before all the test instances. Afterward, we could proceed with the labeling strategy described by Jimenez et al. [54]. Namely, we marked the training instances as “*exploitable*” if and only if they were exploited before the *training date*, which corresponds to the **latest publication date** among all the training instances, while we marked as “*exploitable*” the test instances if and only if they were exploited before the reference date of round R_i .

3.4.4 Model Training & Testing. At each validation round R_i , all the vulnerabilities in the i -th training and test sets were linked with the content of each of the seven corpora (explained at the beginning of Section 3.3)—hence, forming seven “variants” of the i -th training and test sets.

The five pre-trained LLMs, and the four baseline classifiers were trained and tested at this stage without any other processing. On the contrary, the 72 machine learning configurations required further processing according to the selected feature representation schema and data balancing algorithm. Consequently, for each variant of the i -th training set, we fit the four feature representation techniques selected (Section 3.3), i.e., word counting (BoW), term frequency (TF), term frequency-inverse document frequency (TF-IDF), and DOC2VEC (DE). For the first three schemas, i.e., BoW, TF, and TF-IDF, we built three *document-term* matrices, where the rows represent each instance, and the columns represent all the tokens (using the white space as separator) found in the textual content associated. The values inside each cell are weighted depending on the specific schema:

- i BoW assigns to the ij -th cell the number of times the j -th term appears in the i -th document;
- ii TF assigns to the ij -th element the number of times the j -th term appears in the i -th document, divided by the total number of times the j -th term appears in the corpus;
- iii TF-IDF assigns to the ij -th element the TF value multiplied by the IDF (inverse document frequency) of the j -th term, which is computed as the logarithm of the total number of documents divided by the documents where the j -th term appear, hence lowering the weight for terms appearing in too many documents.

Therefore, each instance was represented as a numeric vector, which we used to represent the associated vulnerability. We observe that the final number of features varies among the seven corpora as they have different terms appearing in them; thus, each of the seven variants of the i -th training set ended up having different dimensionalities. On the other hand, the fourth schema, i.e.,

DE, learns a predetermined number of features extracted via the use of a neural network, which learns how to represent all the documents in an unsupervised manner. We chose the *Distributed Bag-of-Words* (PV-DBOW) variant, setting the size of the embeddings to 300, following the configuration that provided the best results in related work [45] and keeping all the other settings to the default recommendations for DOC2VEC. At this point, the two groups of features (tokens and embeddings) extracted from the i -th training set were reused as-is to determine the feature space of all the corresponding seven “variants” of the i -th test set without any modification to avoid data leakage [6]. To do this, on the one hand, we added the test instances into the existing document-term matrices fitted at the training time and weighed the new instances with BoW, TF, and TF-IDF; on the other hand, we fed the fitted DOC2VEC model with the test instances to obtain their embedding in the same space learned at the training time. Lastly, all variants of the i -th training set obtained so far were balanced with two algorithms, i.e., SMOTE over-sampling and NEARMISS (version 3) under-sampling.

3.4.5 Performance Assessment. Once we have obtained the predictions of all the 567 models on the test sets across all the 22 validation rounds, we derived the *confusion matrices* reporting the True/False Positive and True/False Negative predictions. From them, we computed the performance metrics commonly adopted for the binary classification task, i.e., accuracy, precision, recall, and F-measure [77]. The F-measure represents an aggregation of precision and recall, both crucial for evaluating binary classifiers [8]. The trade-off between such values is particularly tricky in the context of exploitability prediction, as practitioners might wish for higher precision to identify the potentially exploitable vulnerabilities correctly but also for high recall to avoid false negatives—i.e., vulnerabilities considered safe but exploitable. However, the F-measure does not consider the number of true negative instances, i.e., the neutral vulnerabilities correctly classified as “neutral”. The problem of exploitability prediction is highly imbalanced, so we were interested in evaluating all four quadrants of the confusion matrix. To this end, we also involved Matthews’s Correlation Coefficient (MCC) [68], which represents an indicator of the correlation between the predicted values and the actual labels of the instances, taking into account the class imbalance in the test set—differently from other traditional metrics.

We computed the selected performance metrics on all the 22 time-aware validation rounds. Consequently, the models had 22 scores of a given metric, which did not allow for direct comparison. Hence, we carried out two kinds of analyses. First, we aggregated the results observed in all 22 iterations of the time-aware validation using a **weighted average**, assigning a weight proportionate to the size of the training set used in an iteration. In other terms, the 22 scores were not treated equally, as (i) the initial iterations faced a problem that is less representative of today’s situation, and (ii) the amount of data the model worked with in the initial iterations was lower. Assigning equal weights to all the iterations, like the simple average, would have provided unrealistic and inflated results, rewarding the models that behaved well in most iterations rather than in the most recent—and, therefore, significant for today’s practitioners—ones. We exploited the aggregated scores to depict the box plots of each of the seven corpora, to highlight the distribution of the performance of the models trained and tested using a given corpus. Besides, we leveraged the Friedman test [40] to discover whether the seven distributions exhibit statistically significant differences ($\alpha = 0.05$). In case a difference is observed, we conducted the Nemenyi post hoc test [73] to identify the pairs of corpora having noticeable differences—indeed, the null hypothesis states that the compared groups have the same distribution. Such a test is robust to repeated comparisons and does not require the tested distributions to be normal. All of this was needed to answer **RQ₁**. Afterward, we plot how the model performance varied over the 22 iterations, having the validation rounds on the x axes and the value scored with a given performance metric over the y axes. Such

plots allowed us to observe the models' general "trend" from different perspectives. Analyzing the trends was needed to answer both **RQ_{2.1}** and **RQ_{2.2}**.

The raw results of our analyses are available in the online appendix of this paper [49].

3.5 Implementation Details and Experimental Infrastructure

The entirety of our experimentation was implemented with a collection of PYTHON scripts. The traditional machine learning algorithms and the baseline models used the implementation provided by SCIKIT-LEARN,²⁰ while the pre-trained LLMs were downloaded from HUGGINGFACE²¹ using its TRANSFORMERS library. In this respect, the exact pre-trained model versions we used are distilbert-base-uncased, albert-base-v2, xml-roberta-base, codebert-base, and codeberta-small-v1, all implemented with PYTORCH.²² The document-term matrices and the feature weighting for BoW, TF, and TF-IDF were done using SCIKIT-LEARN,²³ while the DOC2VEC model was provided by the GENSIM library.²⁴ The data balancing algorithms, i.e., SMOTE and NEARMISS, were implemented with IMBALANCED-LEARN²⁵ library. All the performance metrics relied on the implementation provided by SCIKIT-LEARN, while the statistical tests leveraged the SCIPY²⁶ and SCIKIT-POSTHOCS²⁷ packages.

We ran the experiments involving the baseline models and machine learning algorithms on a Linux machine equipped with a quad-core 1.50 GHz processor and 32 GB of memory. The full data collection procedure took about 11 days, while the dataset cleaning, splitting, and labeling required about 13 hours. Due to the large size of the dataset and the high number of configurations to evaluate and rounds to execute, the models' training and testing phases were considerably time- and resource-consuming, taking a total of 65 hours to complete. To experiment with LLMs, we leveraged a GPU-equipped machine via the VAST.AI²⁸ cloud GPU rental service. The GPU was an NVIDIA RTX 3090 with 24 GB of memory, and the execution of the experiments took about 13 days to complete.

We warmly encourage replication and verification of our work. Thus, we make all the scripts available in the online appendix of this paper [49].

4 ANALYSIS AND DISCUSSION OF THE RESULTS

In this section, we present the results obtained in our experiments to answer our research questions (presented in Section 3.1).

4.1 The Impact of Early Data Source Combinations (RQ₁)

Figures 4 and 5 show the distribution of the aggregated F-measure and MCC scored by the 77 models built on the seven corpora involved in the analysis for **RQ₁**. We can immediately observe interesting differences among the distributions. The models trained using the ⟨BT⟩ corpus had the worst performance, scoring less than ~0.15 median weighted F-measure and less than ~0.10 median weighted MCC. On the contrary, ⟨CVE⟩ and ⟨SF⟩ obtained better performance ($p = 0.001$), reaching ~0.40 and ~0.30 median weighted F-measure, while scoring ~0.25 and ~0.16 median weighted MCC,

²⁰SCIKIT-LEARN website: <https://scikit-learn.org/>

²¹HUGGINGFACE website: <https://huggingface.co/>

²²PYTORCH website: <https://pytorch.org/>

²³Textual features extraction with SCIKIT-LEARN: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

²⁴DOC2VEC with GENSIM: <https://radimrehurek.com/gensim/models/doc2vec.html>

²⁵IMBALANCED-LEARN website: <https://imbalanced-learn.org/>

²⁶SCIPY website: <https://scipy.org/>

²⁷SCIKIT-POSTHOCS documentation: <https://scikit-posthocs.readthedocs.io/en/latest/>

²⁸VAST.AI website: <https://vast.ai/>

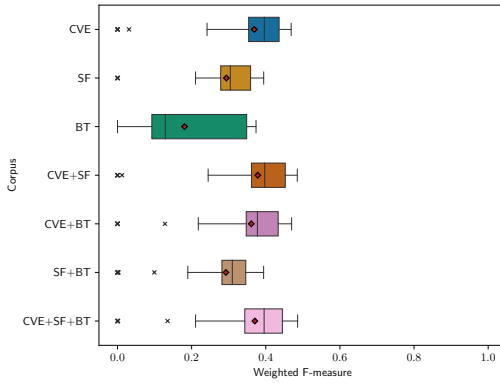


Fig. 4. Weighted F-measure scores obtained by the 77 learning configurations on the seven corpora.

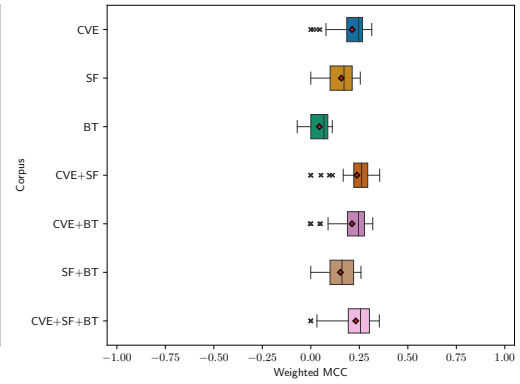


Fig. 5. Weighted MCC scores obtained by the 77 learning configurations on the seven corpora.

respectively. According to the Nemenyi test, the difference between the two corpora is statistically significant for both metrics ($p < 0.05$ for both metrics).

Then, we observe that combining data from multiple corpora led to improvements compared to individual ones. Specifically, adding the text data from the $\langle \text{CVE} \rangle$ corpus to the $\langle \text{BT} \rangle$ corpora can lead up to ~ 0.20 median improvement in both weighted F-measure and MCC ($p = 0.001$ for both metrics). A smaller median improvement, though still statistically significant according to the Nemenyi test ($p = 0.001$ for both metrics), is observed when adding $\langle \text{CVE} \rangle$ to the $\langle \text{SF} \rangle$ and $\langle \text{SF} + \text{BT} \rangle$ corpora, namely slightly less than 0.10 in both weighted F-measure and MCC. Conversely, adding the data from the $\langle \text{SF} \rangle$ or $\langle \text{BT} \rangle$ corpora to the $\langle \text{CVE} \rangle$ corpus does not lead to any noticeable change, as also confirmed by the Nemenyi test ($p > 0.05$ for both metrics). Interestingly, although with minimal (less than ~ 0.01) and no significant differences ($p > 0.05$ for both metrics), the models trained on the $\langle \text{CVE} + \text{SF} \rangle$ corpus experienced a small drop in the median performance for both weighted F-measure and MCC when $\langle \text{BT} \rangle$ is added. In the end, $\langle \text{CVE} + \text{SF} \rangle$ and $\langle \text{CVE} + \text{SF} + \text{BT} \rangle$ have been found to be the best corpora on which the models should train, reaching up to 0.48 weighted F-measure (0.40 on a median) and 0.35 weighted MCC (0.26 on a median). Yet, we cannot confidently determine which of the two is the best option since the models obtained comparable performance with negligible and non-statically significant differences.

The weighted precision and recall (Figures 6 and 7) help better comprehend the F-measure scores observed. The precision distributions are mainly centered around 0.43, though their variance appears higher when the data from the $\langle \text{CVE} \rangle$ corpus is not involved. In other terms, the central tendency seems only slightly affected by the textual content used to describe the instances, but the same does not happen for the variance—i.e., without data from the $\langle \text{CVE} \rangle$ corpus, the models behave largely differently in terms of precision. The best results were obtained by the models trained on the $\langle \text{CVE} + \text{SF} \rangle$ corpus, reaching 0.46 on a median. The situation is somehow different when looking at the weighted recall (Figures 6 and 7). The distributions are noticeably different, with much wider variances; this means that the recall metric is highly subject to the specific model rather than the kind of textual data used. The most “contradicting” results were seen when the text from $\langle \text{BT} \rangle$ corpus is involved, where the median weighted recall is noticeably lower than the mean, and the boxes are wider. Such a scenario indicates the presence of many models having very low recall, i.e., models tending to avoid predicting true (i.e., “exploitable”), and models that predominantly predicted true that can easily raise their recall. The models that did not use the

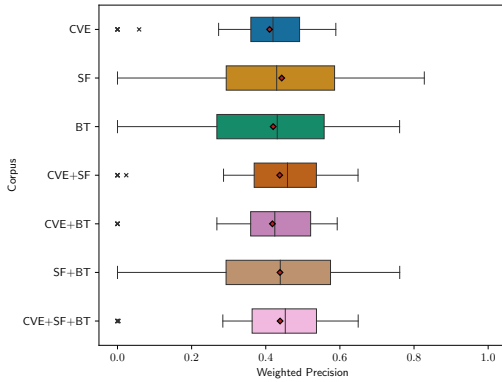


Fig. 6. Weighted precision scores obtained by the 77 learning configurations on the seven corpora.

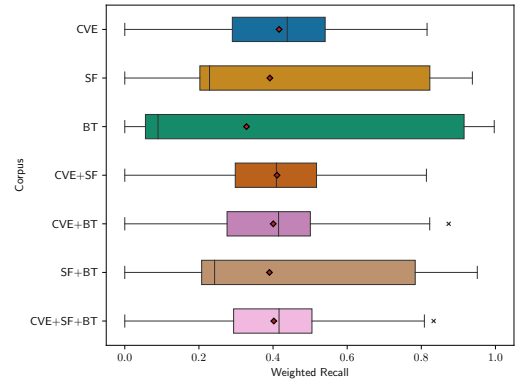


Fig. 7. Weighted recall scores obtained by the 77 learning configurations on the seven corpora.

⟨CVE⟩ corpus scored less than 0.25 weighted recall on a median. Once adding ⟨CVE⟩, the central tendency between the corpora becomes more equalized ($p > 0.05$).

All these results indicate that the sole CVE description is sufficient for determining the majority of the performance [4, 45]. The text from SECURITYFOCUS can provide additional information that further boosts the performance without changing the general trend. Despite not giving useful information on its own, the text from BUGTRAQ does not hinder the predictions when mixed with text from other sources.

👉 **Answer to RQ₁.** The text from the experimented data sources significantly impacts the model performance, affecting up to 25% and 15% of the median weighted F-measure and MCC metrics, respectively. The central tendency of the models' precision is essentially unaffected by the specific corpus selected, but not the variance, which is wider when the text from ⟨CVE⟩ corpus is not involved. Wide distributions are also observed for the recall, irrespective of the corpus. Involving textual data from CVE always leads to improvements in all perspectives, which is further improved if the data from SECURITYFOCUS are added as well. The text from BUGTRAQ alone does not inform the models adequately but can be added as an extra source along with CVE and SECURITYFOCUS without harm. In the end, the best corpora for training the models are ⟨CVE + SF⟩ and ⟨CVE + SF + BT⟩, showing no relevant differences between the two.

4.2 The Performance of Different Learning Configurations (RQ₂)

Once we determined the best corpora to train the prediction models, we investigated the performance scored by the experimented learning configurations to answer RQ₂. We subdivided it into RQ_{2.1} and RQ_{2.2} to have focused analyses on the models built with the traditional learning pipeline and those leveraging end-to-end pre-trained LLMs. For this analysis, we chose to focus on the models trained and tested on the ⟨CVE + SF⟩ corpus as its content determined the best models overall. The raw results for the other corpora can be found in our online appendix [49].

To answer RQ_{2.1}, we analyze the ML models built with the traditional pipeline, which is made of three key parts: (1) feature representation, (2) training data balancing, and (3) learning algorithm. Figure 8 provides a broad overview of the F-measure scores obtained by the six learning algorithms on the 12 training settings made by the combination of the four feature representation schemas and the three data balancing algorithms. We observe that all models under every training setting

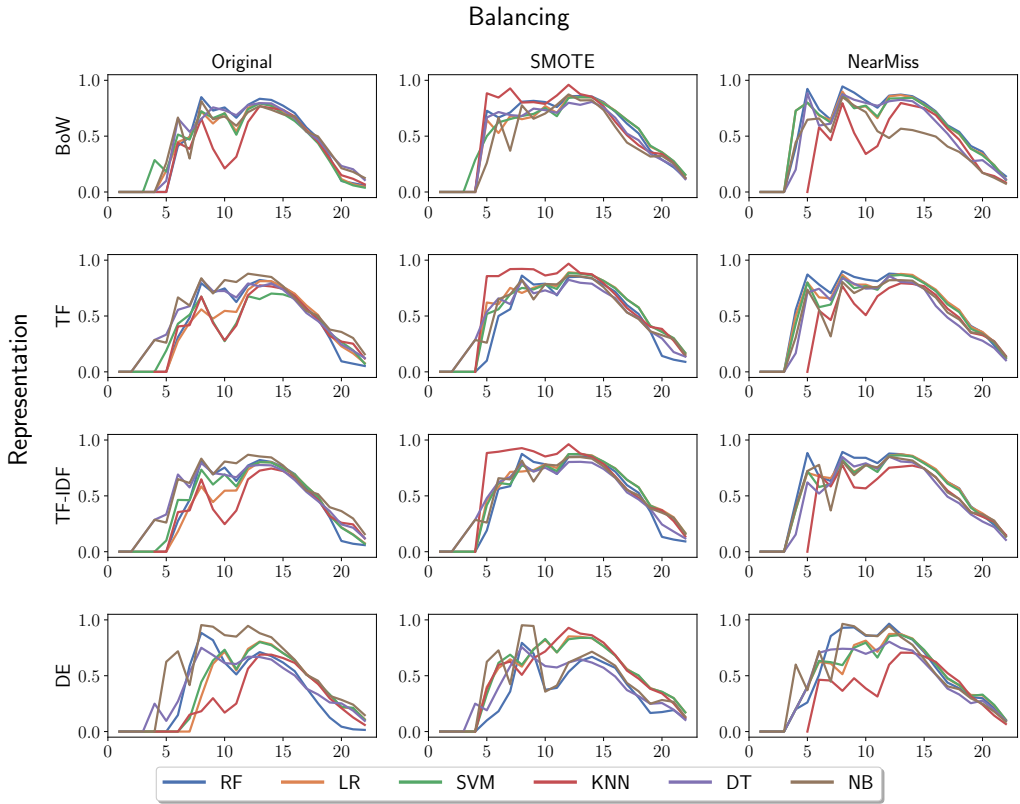


Fig. 8. F-measure scores obtained in the 22 time-aware validation rounds by the 72 traditional ML models on the (CVE + SF) corpus. Each subplot shows the six learning algorithms, each with a unique color. The subplots in the rows share the same feature representation schema, while the subplots in the columns share the same balancing algorithm for training data.

followed one great pattern: the F-measure steadily increases—net of sporadic drops—from the 1st round to the 12th, i.e., the point where almost all models achieve the best score of 0.97. Yet, all models start dropping their performance from that round on, reaching their lowest peak of less than 0.16—excluding the very initial rounds. This phenomenon shows that the most recent “versions” of such models are not able to recognize exploitable vulnerabilities properly, despite seeing dozens of thousands of examples during training. It seems the learning becomes less and less fruitful as the round goes by, likely due to the difficulty of recognizing a clear distinction between exploitable vulnerabilities and those not exploited yet, among many examples. In other words, the text data were enough to recognize the exploitability of “historical” vulnerabilities but are less helpful for modern-day vulnerabilities. It is worth pointing out that there could be other reasons behind such a drop. For instance, we observe that the disclosure of public exploits has become less frequent than it used to be in the past, from over 15,000 in the period 2001-2010 to less than 7,500 in the period 2011-2020. This difference becomes even more relevant when looking at the number of disclosed vulnerabilities: around 42,000 in 2001-2010 and around 100,000 in 2011-2020. Thus, the number of disclosed vulnerabilities doubled in a decade while the published exploits halved. This inevitably

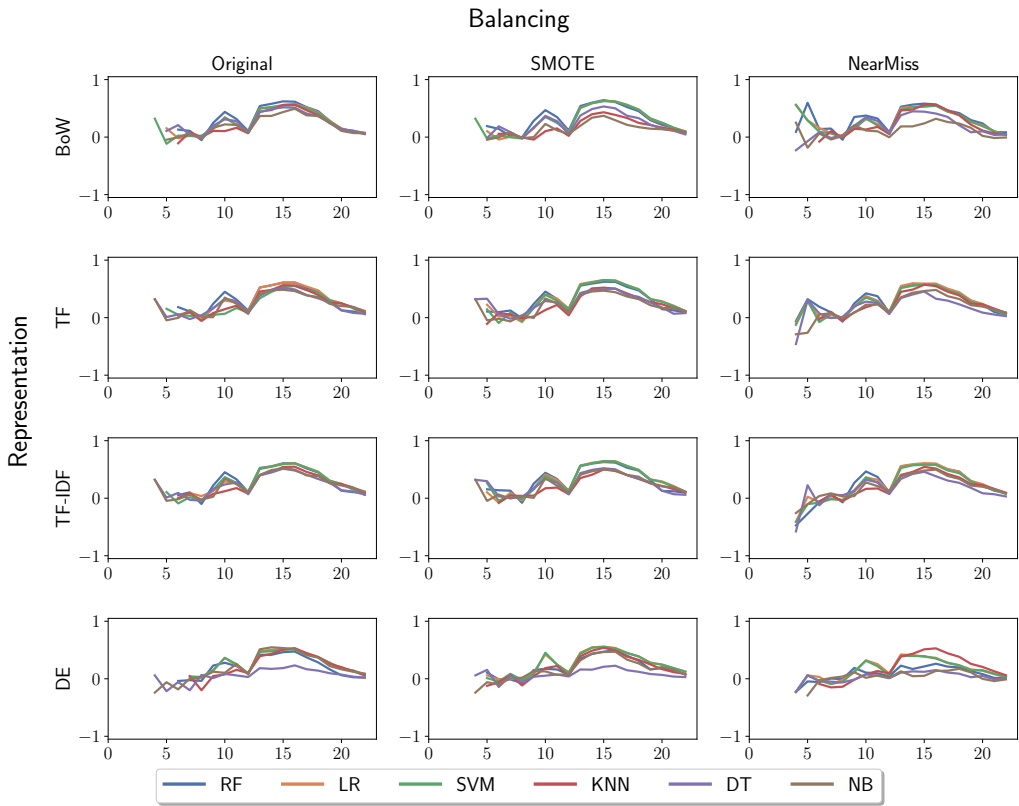


Fig. 9. MCC scores obtained in the 22 time-aware validation rounds by the 72 traditional ML models on the $\langle \text{CVE} + \text{SF} \rangle$ corpus. Each subplot shows the six learning algorithms, each with a unique color. The subplots in the rows share the same feature representation schema, while the subplots in the columns share the same data balancing algorithm for training data.

affected the distribution of true and false instances, ending up with highly imbalanced test sets in the latest validation rounds.

Due to the limited reliability of F-measure for measuring the model performance when the number of true instances is noticeably lower than the number of false instances [99], we also looked at the MCC metric (depicted in Figure 9) to observe whether a similar pattern occurred. Interestingly, the models achieve an MCC score around zero in the 12th round, indicating the absence of any correlations between the model predictions and the target variable (i.e., the exploitability). A lack of correlation means that the models make utterly unrelated predictions with the target variable, implying that the model performs no better than a fully random or constant classifier [9, 99]. The diverging results of MCC and F-measure scored at the 12th round suggests that many models in that round behaved almost like a constant classifier always predicting true; this behavior benefited the F-measure since over 95% test instances had true label in the 12th round but not the MCC metric, which does not reward models making one-way predictions.

The best MCC scores were obtained around the 15th round, going beyond 0.60 MCC in the best-case scenario, i.e., when SMOTE balancing is employed. Such a score indicates the presence of a strong positive correlation, which suggests the model performs well. This is further confirmed

by the good F-measure scores, reaching 0.80 when SMOTE is used. Thus, the 15th round provides a definitely better trade-off than the 12th. All the models in the 15th round were trained on all vulnerabilities disclosed until December 2008 and tested on those disclosed until August 2011. We observe that the amount of true and false test instances is more balanced (around 50% for both), indicating that the exploitability prediction task was easier than it is today—indeed, in the last round the number of true instances in the test set is just the 5%. Unfortunately, after this round, all the models meet a similar demise seen for the F-measure: they slowly converge to no more than 0.12 MCC in the last round.

Looking deeper at the effect of feature representation schema on the F-measure trends (Figure 8), we observe that the models based on the document-term matrix (i.e., BoW, TF, and TF-IDF) share the same general trends once a data balancing technique is applied. In particular, we observe that the effect of a balancer looks the same in all three schemas, favoring and hindering the same classifiers. For instance, the KNN classifiers draw many benefits from an oversampled training set (i.e., SMOTE). Moreover, all the classifiers follow closely similar trends when SMOTE is used. On the contrary, the models have highly diversified trends with document embeddings (i.e., DE), standing out from the other three feature representation schemas. The KNN classifier still can be seen benefiting from the use of SMOTE, though with inferior performance than in other schemas. The MCC trends (Figure 9) exhibit a similar effect though with less diversification, i.e., the effect of the feature representation schema and the data balancing is smoother, particularly with TF and TF-IDF. Interestingly, the training sets undersampled with NEARMISS determined models with negative MCC scores (reaching less than -0.3 in several cases), indicating the presence of moderate negative correlations between the model predictions and the target variable; yet, this happened only in the initial validation rounds, which does not imply any negative impact of this balancer.

We used the weighted metrics to determine the models that achieved the best results across all rounds. We found that the best model overall was a LOGISTIC REGRESSION classifier (LR) using TF feature schema and with a training set oversampled by SMOTE, scoring 0.49 weighted F-measure and 0.36 weighted MCC and touching 0.82 F-measure and 0.65 MCC in the 15th round. In particular, we observed that most learning algorithms had their best F-measure and MCC scores with TF and SMOTE. The story is slightly different by looking at the precision and recall. We found that the learning algorithm with the highest weighted recall was KNN, reaching 0.80 with BoW and SMOTE—0.08 higher than the score obtained with TF and SMOTE. Symmetrically, the RANDOM FOREST (RF) achieved the highest weighted precision score, reaching 0.65 with TF-IDF without data balancing—only 0.04 higher than the score obtained with TF and SMOTE. In the end, we observed a generally positive trend with TF and SMOTE, though maximizing a specific metric might require a specific learning configuration.

Lastly, we looked at the performance scored by the four baseline models, i.e., the random (RND), the pessimistic (PES), the optimistic (OPT), and the stratified (STR) classifiers. Among the four, the best baseline classifier was PES, achieving 0.37 weighted F-measure and 0.26 weighted precision. We remark that the MCC is always zero as the true and false negatives are always zero, while the recall is always maximum (i.e., one) for the same reason. Such results show that the experimented prediction models do make meaningful predictions as the best model, i.e., the LOGISTIC REGRESSION (with TF and SMOTE), outperforms PES by 0.12 and 0.34 in weighted F-measure and precision, respectively. Nevertheless, PES outperforms all models in the 12th round for F-measure. Indeed, due to the large presence of true instances in the test set, the PES model has a very low probability of making false positive predictions, obtaining high precision in return and boosting the F-measure—thanks to the recall score fixed to one. In any case, its performance drops in all the rounds, where the test instances have less imbalanced distributions.

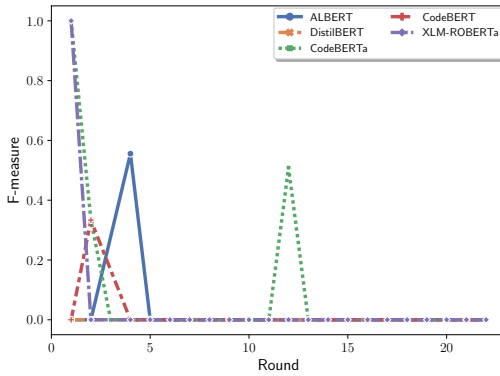


Fig. 10. F-measure scores obtained in the 22 time-aware validation rounds by the 5 LLMs on the ⟨CVE + SF⟩ corpus.

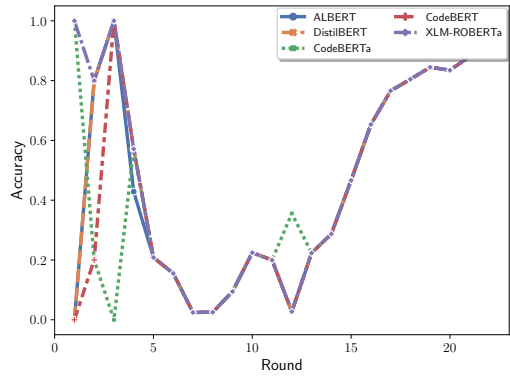


Fig. 11. Accuracy scores obtained in the 22 time-aware validation rounds by the 5 LLMs on the ⟨CVE + SF⟩ corpus.

✦ **Answer to RQ_{2.1}.** All traditional ML models had their F-measure increase until the 12th validation round, touching the peak of 0.97, until dropping to 0.16 in the latest round. The MCC scores followed a similar trend, despite the growth arriving at the 15th round, reaching ~0.60. The 15th round determined the best trade-off among the two metrics. Oversampling with SMOTE generally gives benefits to the models, particularly if applied in conjunction with TF; in this setting, the LOGISTIC REGRESSION classifier achieved the highest weighted F-measure and MCC scores, i.e., 0.49 and 0.36, respectively, outperforming the pessimistic baseline model under all fronts. KNN achieved the highest weighted recall of 0.80 with BoW and SMOTE, while RANDOM FOREST reached 0.65 weighted precision with TF-IDF without any data balancing.

To answer RQ_{2.2}, we analyze the models made with the pre-trained LLMs. We found that, with the exception of a few sporadic rounds, all models tend to act like perfect optimistic classifiers, i.e., always predicting “neutral” (having false label). Therefore, the F-measure (Figure 10) turned out to be extremely low (the weighted aggregated score did not go beyond 0.01 with CODEBERTa) as a consequence of the recall being always zero—due to the absence of any true prediction). Such behavior had an extremely positive impact on the accuracy (Figure 11), on which all the models achieved very high performance as the rounds went on; this happened because of the scarce number of true instances in the test sets of the later rounds.

The round that had the most interesting performance in the ⟨CVE + SF⟩ corpus is the 12th, the same where the traditional ML models achieved the highest F-measure scores. In such a round, the CODEBERTa model reached 0.51 F-measure thanks to the quasi-perfect precision, i.e., 0.98, which happened because of the large number of true instances in the 12th test set that minimized the chances of making false positive predictions. Nevertheless, given the peculiarity of the 12th test set, it is not clear whether in this round CODEBERTa successfully learned something or it was just making random predictions (with 33% true predictions and 67% false ones). Such an interesting behavior did not only happen for the ⟨CVE + SF⟩ corpus but for all the other corpora, though with different “fortunate” rounds.

Ultimately, we can conclude that the pre-trained LLMs could not learn anything from the training phases—except for the few “fortunate” rounds—despite the large amount of data available. The only models that apparently learned something were CODEBERTa and CODEBERT, both having

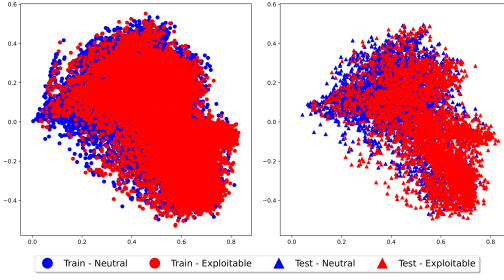


Fig. 12. Visualization of the 15th training and test instances of the ⟨CVE + SF⟩ corpus represented with TF in a 2-dimensional space with LSA.

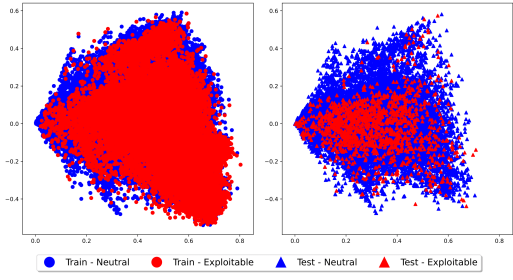


Fig. 13. Visualization of the 22nd training and test instances of the ⟨CVE + SF⟩ corpus represented with TF in a 2-dimensional space with LSA.

experienced source code text during their pre-training stage; yet, they both tend to behave like the other models in later rounds. We believe the reasons could be imputed to the lack of a massive pre-training on text containing the typical vocabulary of the security domain, but also to an inadequate data preprocessing for the experimented models.

👉 **Answer to RQ_{2.2}.** The pre-trained LLMs used as-is are inadequate for assessing the exploitability leveraging early data, behaving like constant classifiers always predicting false. There are few exceptions in certain rounds of the time-aware validation, but whether the predictions were made randomly is unclear.


4.3 Further Analysis

The time-aware validation setting allowed us to observe how the models behave at different points in time where the training and test sets had diversified compositions. We wanted to shed light on the composition of both training and test sets to comprehend the possible reasons for the model to make such predictions further. Thus, we employed a dimensionality reduction technique based on the Singular Value Decomposition (SVD) [33], which projects the data into a lower dimensional space using matrix factorization. Such a technique is better known as *Latent Semantic Analysis* (LSA, a.k.a. Latent Semantic Indexing, LSI) [27, 31] when adapted for highly sparse data, like the textual represented with BoW, TF, and TF-IDF. Essentially, this technique forms a lower-dimensional “semantic space” of a given size—typically vastly lower the number of terms—where the instances sharing similar concepts are mapped to the same cluster, also dealing with cases of synonymy and polysemy of terms.

In our case, we chose to build a semantic space of two dimensions to allow plotting into a 2D space and inspect how the training and test instances are distributed. Specifically, we focused on the training and test sets employed in the most interesting rounds that emerged from the model performance analysis. Therefore, we inspected the 15th and the 22nd rounds due to their contrasting performance; the former achieved the best trade-off between F-measure and MCC, and the latter had the worst performance overall. In continuity with the previous analyses, we focused on the ⟨CVE + SF⟩ corpus and opted for visualizing the document-term matrices made with TF schema as it was the schema that had the best results overall.

Figures 12 and 13 show the scatter plots of the training and test instances drawn from the 15th and the 22nd rounds, respectively. Both plots depict the large number of instances involved in training and testing. We immediately observe in all cases, the “exploitable” (true) and “neutral” (false) instances are somehow “intermixed”. This could explain why many models had trouble

understanding the difference between the two classes of instances and so opted to behave like constant classifiers in several cases. Nevertheless, we cannot exclude the visualization algorithm that failed to faithfully preserve the differences among the instances, though it is recommended to visualize data with textual features. Such an aspect is worth further investigation. Looking deeper at the training and test sets of the 15th round, we observe the two share a similar arrangement. This might indicate that the models, once trained, did not find a different problem once going to the test phase, which might be the main motivation for the good performance obtained in that round. On the contrary, the “neutral” (false) training and test instances of the 22nd round share the same arrangement but the “exploitable” (true) instances do not. Indeed, the arrangement in the test phase seems like a subset of the arrangement seen at the training time. Likely, this could be one of the reasons why the models increased their false positive rate (i.e., false instances deemed as true) due to this reduced presence of true instances at the testing time.

 **Further Analysis Summary.** There is a noticeable discrepancy between the arrangement of training and test instances in the 22nd round, where all models’ performance dropped to their minimum. Such a discrepancy is not observed in a good round like the 15th. Visualizing the composition of training and test instances during the validation rounds seems an interesting diagnostic tool to find the possible causes behind misclassification.

5 DISCUSSION AND IMPLICATIONS

The results achieved in our study shed light on several aspects that may lead to several implications for the research community and the practitioners, as discussed in the following.

Searching for a Reliable Ground Truth. The results reported in Section 4 revealed the noticeable performance drop that affected all the models—including the baselines such as the pessimistic classifier—as the validation rounds proceeded. In this respect, we made two key observations: (1) the F-measure score is directly proportionate to the number of true instances (i.e., “exploitable”) appearing in the test set, independently from the composition of the training set; (2) the MCC scores revealed the existence of several rounds with positive correlations between the model predictions and the target variable. Similar findings were only encountered in similar research work applying a time-aware validation framework [2, 17]. Yet, such works only brought attention to the aggregate score performed in all rounds, while we opted for a hybrid strategy, presenting both the aggregated (weighted) scores and focused attention to those rounds exhibiting particular behaviors. We suspect that one of the main reasons behind such results lies in the strategy adopted to build the ground truth. In this work, we relied on the EXPLOIT DATABASE because of its good reputation and popularity among researchers in exploitability prediction [2, 34, 50, 85]. Nevertheless, we observed a noticeable reduction in the publication rate of exploited vulnerabilities—i.e., half as many exploits released in the period 2011–2020 than in the previous decade, with the rate of disclosed vulnerabilities doubled. Indeed, it seems that it has been struggling to keep up the pace of newly disclosed vulnerabilities in recent years. This could imply that either exploits are disclosed with less frequency than before or the rate of new vulnerabilities is too high to keep up the pace; this phenomenon makes the EXPLOIT DATABASE progressively less reliable for building a solid ground truth in both cases. In this respect, our study constitutes a baseline for future re-evaluations with alternative data sources to build better ground truths [2, 34]. Indeed, many other sources point to instances of exploits (or tentative exploits) observed in the wild. For example, SYMANTEC ATTACK SIGNATURES collect traces of attackers’ attempts via intrusion detection systems.²⁹ In the context

²⁹SYMANTEC ATTACK SIGNATURES: <https://www.broadcom.com/support/security-center/attacksignatures?>

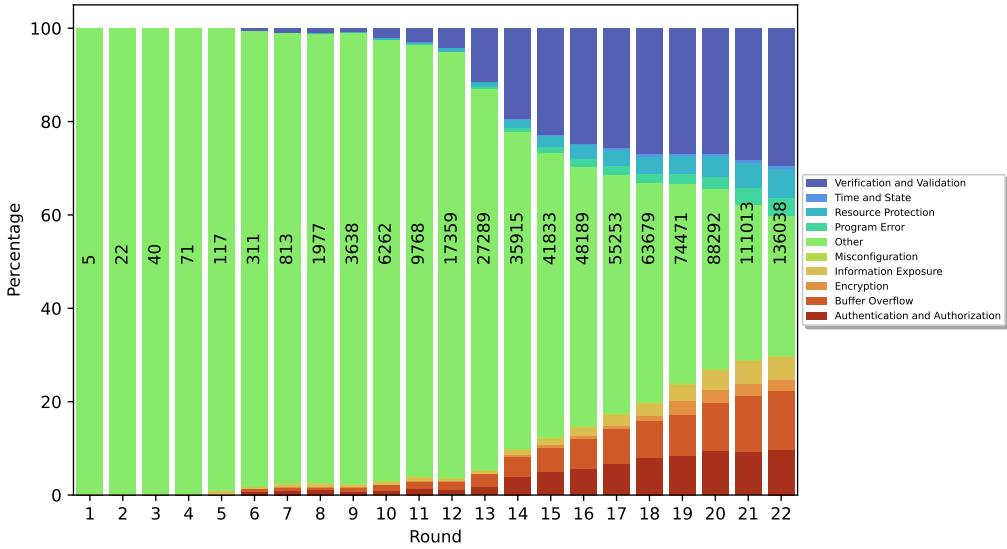


Fig. 14. Distribution of the vulnerability in our dataset over the 22 splits into ten custom categories reflecting their weakness type.

of PROJECT ZERO, Google gathers 0-day exploits observed in the wild, enriched with a detailed root cause analysis.³⁰ The US Cybersecurity & Infrastructure Security Agency (CISA) curates the *Known Exploited Vulnerabilities* (KEV) catalog, containing hundreds reports of exploited vulnerabilities.³¹ Due to their newness, the size of such datasets is still limited (e.g., GOOGLE PROJECT ZERO has only 69 entries as of February 2024), though their increasing popularity should address this problem eventually, making them suitable for large-scale analyses like ours. We envision a *triangulation* of multiple strategies to improve the reliability and quality of the labeling process. To this end, there is a need for novel and automated monitoring solutions that automatically discover “silent” exploits on the web and map them to the related vulnerabilities; thus, the exploitability prediction models could rely on a wider and continuously growing knowledge base about real-world exploits.

Classifying Vulnerabilities for Fine-grained Inspections. Our large-scale analysis involved all disclosed vulnerabilities in NVD until November 03, 2021. In our work, we did not make any difference between vulnerabilities, e.g., analyzing web-based and memory-related vulnerabilities separately, but treating all of them as equal. Many factors concerning vulnerabilities inevitably influence any prediction activity, especially the exploitability prediction. In particular, how the distribution of vulnerability types varies over type could be one of such factors that might influence the prediction performance observed. To reach this goal, we have given each vulnerability a category guided by its assigned CWE. Namely, we re-mapped the given CWE according to the “*Simplified Mapping*” view provided by CWE itself.³² Such a step was meant to greatly reduce the many weakness types into a more reasonable set of categories, allowing us to decrease it from 180 to 89. Being this number still great, we opted to further assign new categories based on our knowledge of the 89 CWE types resulting from the first re-mapping, ending up with ten broader categories like “*Authentication and Authorization*” and “*Resource Protection*”. Figure 14 shows the distribution trend

³⁰Google Project Zero’s 0-DAYS IN-THE-WILD: <https://googleprojectzero.github.io/0days-in-the-wild/>

³¹CISA KEV Catalog: <https://www.cisa.gov/known-exploited-vulnerabilities-catalog>

³²CWE-1003: <https://cwe.mitre.org/data/definitions/1003.html>

into such categories over time (over the 22 splits of the dataset). We can immediately observe that as the year passes, the precise type of vulnerability assigned becomes clearer. Indeed, during the initial periods of the CVE system, most vulnerabilities had their CWE not specified (“NVD-CWE-noinfo” or “NVD-CWE-Other”), resulting in many CVEs falling into the “Other” category, particularly during the first 12 rounds. Such lack of information does not allow us to easily understand whether the vulnerability types could have played a role in the performance drop we observed at the later rounds. The results of this work should be remade into a subset of vulnerabilities for triangulating the issue that affected the model performance.

Engineering the Learning Configuration. As observed in the context of RQ_2 , the four feature representation techniques had a relevant impact on the overall models’ performance. In this study, we relied on widespread settings to set up the text pre-processing pipeline and to configure the DOC2VEC model without carrying out a profound empirical investigation. Indeed, our goal was to assess the key differences among the main techniques employed when working with textual data. Hence, our work does not declare the best feature representation technique on all fronts but rather encourages the evaluation of alternative learning configurations and techniques. The Latent Semantic Analysis (LSA) [27, 31] used to reduce the dimensionality of the document-term matrices (Section 4.3) is a candidate technique that can be employed to represent the textual features the traditional ML models can use, acting somehow similarly to the embedding strategies like DOC2VEC. As regards the *time-aware* validation setting, we followed Liu et al.’s [63] approach by considering a deployment setting in which the knowledge base grows over time. In a different way, the work by Bullough et al. [17] employed a “sliding window” to train only on a limited set of past data, i.e., only those that are temporally closer to the testing data. The rationale of their choice is that recent data might represent the reality better than older data, as some characteristics might have changed over time—i.e., a *concept drift* has occurred [97]. For instance, the style and content of discussions in BUGTRAQ written in 2010 might differ from those of 2000, negatively affecting the models when learning the relations between the textual features and the target variable. The traditional sliding window approach completely ignores older instances during the training based on a pre-determined or moving threshold (i.e., the “window size”). Alternatively, we could also assign a lower *weight* to older instances, using a “decaying window”, so that the learners would give less importance to the old instances that likely induced the models into error. In addition, by employing a mechanism to assess the quality of the online discussions, e.g., their readability [89] or the amount of their informative content [22], we could assign higher weights to “good” instances and instruct the models to give more attention to them during the training, hopefully improving their overall capabilities.

On the Practical Usages of Early and Realistic EPMs. Adopting an *early* exploitability prediction model provides many advantages in assessing the severity of newly discovered vulnerabilities. Let us consider a scenario where a software project adopts one. When a new vulnerability is discovered, either by internals or externals, the developers report the issue to MITRE and request the allocation of a CVE record, where they explain the issue found. Once third-party experts verify the issue, the vulnerability is officially disclosed in a CVE record containing the first official description in natural language. Such a description can be directly fed into the early exploitability prediction model to readily generate an initial assessment of that vulnerability. Besides, if other free commentaries are already available—e.g., via GITHUB issues—the prediction model can integrate those pieces of information to boost the prediction accuracy further, as we also observed during the analyses for RQ_1 . Should the model flag the vulnerability as potentially exploitable, the developers can take specific countermeasures [56, 86] to (i) address the vulnerability earlier than other issues, (ii) release the software version containing the patch quickly, and (iii) adopt a better communication

strategy to recommend the users to install the update as soon as possible. It is worth remarking that any countermeasure adopted in this sense is meant to hasten the *vulnerability remediation* process, not to replace other forms of security assessment like “late” exploitability prediction models or the CVSS analysis. In this respect, *early* assessment can also be used to support the security analysts in charge of making the CVSS measurement, who can rely on an additional “opinion” when it comes to judging the vulnerabilities’ nature. In particular, bringing forward the assessment of just-disclosed vulnerabilities facilitates the *prioritization* of all the security issues found until that moment. Indeed, developers can better understand which issue requires more attention than others and allocate adequate resources accordingly in the hope of reducing the duration of the exposure window and, therefore, the risk of being attacked, as explored by Jacobs et al. with EPSS [52]. Afterward, as soon as new information on the vulnerabilities becomes gradually available, e.g., a Proof-of-Concept is disclosed, developers can progressively leverage more reliable solutions to adjust the prioritization of their interventions, such as EVOCATIO fuzzer [53]. In such a mechanism, we envision that models based on *early* data can represent the first step of a prioritization pipeline which takes advantage of all the strengths of existing solutions as soon as the information they use becomes available. Despite the performance observed at the latest validation rounds not supporting the practical usefulness of such early models, we believe this work acts as a cornerstone for determining the feasibility of *early* vulnerability assessment, willing to channel more attention to this topic and express its potential to the utmost.

Early Predictions and Beyond. Our empirical investigation did not aim to provide a cutting-edge exploitability prediction model but rather to evaluate its performance with early data and in a realistic scenario. We acknowledge the existence of models that achieved better results in the literature [14, 45, 85]; yet, many of them did not consider the precautions indicated by [17] or those we adopted in this work. In this respect, we believe that replicating previous work under a realistic validation setting is necessary to estimate the models’ real effectiveness. Moreover, we envision a combination of all existing models, both *early* and *late*, to develop an *incremental exploitability prediction system*, i.e., an integrated framework that provides the best possible predictions according to the information available at a given time. For instance, after discovering a new vulnerability (day zero), the incremental system would just rely on the short description and the initial online discussions—as we have presented in this paper; on the day the experts make in-depth analyses, the system will consider all the features obtainable from the CVSS vector to further improve its predictive power—acting as “late” models. Such a solution may express its full usefulness in the case of borderline classifications, i.e., when the early predictions fall too close to the decision threshold, making the model unsure about the appropriate class to assign. In such scenarios, the system might recommend waiting for additional data, such as the CVSS exploitability metrics, before providing a more trustworthy response. Furthermore, this framework could be integrated with an *impact prediction* module that estimates the harms that the potential exploitability of that vulnerability could cause to the confidentiality, integrity, and availability of the targeted asset. This additional piece will cover the second part covered by CVSS base metrics, i.e., the “Impact metrics,” fulfilling the role of assisting the human experts in providing a broad understanding of the risks connected to keeping a vulnerability unfixed.

6 THREATS TO VALIDITY

Threats to Construct Validity. We mined the full content of the National Vulnerability Database (NVD) combined with CVE LIST to collect all the known vulnerabilities disclosed before November 03, 2021, being careful to avoid the inclusion of malformed and rejected CVEs. We did not perform an extensive manual validation of the retrieved dataset to detect possible curation errors, such as

a CVE incorrectly pointing to an external reference related to a different vulnerability. However, we are confident that the considered data sources are reliable since both databases are known to maintain high-quality data. Besides, we deliberately focused only on the URLs labeled as *BID* or *BugTraq* for three reasons: (1) URLs of these kind point to well-known sources where developers used to discuss vulnerabilities way before their official disclosure; (2) the pointed websites were easy to mine—a common HTML parser sufficed—to gather the required data; (3) developing a generic script able to mine all the thousands of different websites reachable from the CVEs would have been impractical. We handled the shutdown of both SECURITY FOCUS and BUGTRAQ using the WAYBACK MACHINE service and the SECLISTS archive to recover the missing links with the CVE records. Nevertheless, we cannot guarantee the freedom from missing or incorrect links caused by WAYBACK MACHINE or the imprecision of the pattern matching heuristic employed to reach the right page on SECLISTS. In any case, the approach of early prediction models is not strictly bound to *BID* and *BugTraq* references, and it can be adapted to any other source of online discussions with just minor tweaks.

The text of the online discussions contained many irrelevant data, such as e-mail addresses, PGP signatures, and hex numbers. We applied regular expressions to capture these patterns and remove them to improve the quality of the document corpus. In addition, we adopted the recommended pre-processing steps when working with natural language text to allow the feature representation techniques to learn a compact and representative vocabulary. We are aware that our text cleaning procedure may not have been complete and could have left other forms of noise, such as partial code snippets; yet, to the best of our knowledge, there are no tools able to capture partial code elements for any programming language; hence we opted not to implement an ad-hoc solution as it would have required dedicated effort and extensive validation.

When assigning the labels to the instances in our dataset, we carefully avoided labeling all instances outside the context of the *time-aware* validation. To this end, we followed the strategy proposed by Jimenez et al. [54], assigning more realistic labels to the instances at each round. Specifically, we labeled as “exploitable” (true class) the instances in the training set that were exploited before the *training date*—i.e., the latest publication date in the training set—and we marked as “exploitable” the test instances only if they were exploited before the date of the last vulnerability published in that round.

Threats to Internal Validity. The investigation for RQ_1 analyzed the impact caused by the seven corpora created from the three data sources considered, i.e., CVE, SECURITY FOCUS and BUGTRAQ, and their combination. The combination consisted of applying a string concatenation to create the four combined corpora before creating the document-term matrices or fitting the WORD2VEC model—this determined different feature spaces for each corpus. Not only did the analysis help understand the impact of each corpus, but it also showed that text-driven early prediction models can be employed with any source available, though with noticeably different performance. In other words, it is not mandatory that a vulnerability is disclosed via CVE before running the predictions, but the entire procedure can be done with any kind of text explaining the issue.

As indicated by Bullough et al. [17], several exploitability prediction models in literature had some issues in their machine learning setup. First, we adopted a *time-aware* validation setting to simulate a realistic production scenario in which the prediction model is periodically re-trained and deployed. We deliberately avoided a fully-random cross-validation as our data had time relations among them; indeed, training on “future” data to predict data belonging to the “past” would have generated inflated and misleading results. Moreover, we were careful to avoid applying the feature encoding and data balancing (where applied) on the test data, but only on the training set made up at each iteration of the *time-aware* validation. Indeed, such bad practices would produce overly

optimistic results, as the models would learn information from data that should be left completely unseen before the testing phase because they represent instances to predict in a real deployment scenario [6].

To perform the *time-aware* validation, we split the dataset into several folds, each made of the vulnerabilities disclosed in (about) one year and a half time span—precisely, 532 days. We started the splitting from the last published CVE in 2021 and “jumped” back in time to form the 22 folds. The size of such a time span was determined by the 90th percentile of the exploitation time distribution, i.e., the duration of the *uncertainty window*. We made this choice to observe how the models behave when the uncertainty window is made of totally different sets of vulnerabilities. We acknowledge that there exist different ways to create the folds, such as by equally splitting the dataset by the number of CVEs. The results we obtained are still subject to the choice we made to set the size of the uncertainty window. We chose the 90th percentile of the exploitation time distribution as it represents a largely sufficient time to let exploits emerge. As a matter of fact, the average exploitation time, i.e., 194 days, appeared quite limited and too eager. We are aware of different, and perhaps more appropriate, widths of the uncertainty window that could determine more valid results, and that would be worth exploring with dedicated further analyses.

When generating the document embeddings from the training corpora with DOC2VEC, we used the configuration that provided the best results in previous work [45] and others settings recommended for DOC2VEC models, e.g., setting to 300 the size of the embedding or using the *Distributed Bag-of-Words* variant. The results achieved by this feature representation technique might change if different configurations are employed.

Threats to External Validity. We used the EXPLOIT DATABASE as the main source to build our ground truth (i.e., to label the CVEs with `true` and `false`) because of its reliability and completeness. Nevertheless, it only stores Proof of Concepts (PoC) and exploits publicly released by their authors without tracing any exploit observed in the wild (e.g., via attack signature detection) as done by SYMANTEC ATTACK SIGNATURES or Google Project Zero’s 0-DAYS-IN-THE-WILD (described in Section 5). Therefore, our models can only predict whether an exploitation will be released without generalizing to other forms of exploitation. Moreover, the observed results hinted at possible flaws in our ground truth that caused the model performance degradation. Thus, integrating multiple data sources could improve the quality of the ground truth and, hopefully, the performance as well.

The exploitability prediction models experimented in this work target disclosed vulnerabilities. This choice was driven by the fact that the metadata for such vulnerabilities is available for initial assessments. Hence, the models cannot estimate the exploitability of 0-day vulnerabilities since they are supposed to be unknown to the developers or any other party involved in taking care of the system’s security. Predicting the risk of undergoing 0-day exploits inevitably might require monitoring the accesses to the application or any other suspicious actions, relying on principles different from those recalled in this work [44]. Nevertheless, the prediction of exploits to known and disclosed vulnerabilities can also be used as a proxy indicator for estimating the exploitation of other unknown vulnerabilities in the system sharing commonalities with those already disclosed [13]. Furthermore, the experimented models are meant to predict the event of future exploitation for individual vulnerabilities, in line with all the related work presented in Section 2. Hence, the models cannot predict attacks concerning multiple vulnerabilities or chains of exploits. Achieving such a goal is indeed feasible, though it might require more mature models to make accurate predictions for individual vulnerabilities.

Threats to Conclusion Validity. From all the 504 models, we computed multiple metrics capturing the classifiers’ performance from different points of view, reducing the risk of drawing erroneous conclusions. In particular, we deliberately did not consider the accuracy—except for observing the

scarce performance of LLMs—as it produces largely inflated results leading to optimistic conclusions in imbalanced problems. In this paper, we largely relied on the F-measure and MCC metrics to observe how the model performed. The rest of the raw results are in the online appendix [49].

We aggregated the results of the 22 validation round with the weighted average to have a single number representing the overall performance and facilitate the comparison. We preferred this aggregator to other popular choices, such as the simple average or the median, because the 22 validation rounds are not equivalent representations of the same problem. Indeed, in the 15th and 22th rounds, all models achieved utterly different performance; nevertheless, the exploitability prediction problems of those two rounds cannot be directly compared as they are separated by the events that occurred in ten years. Therefore, we assigned more weights to the rounds having wider training sets. We also looked at the aggregated scores obtained using the simple average and the median. For instance, the model that had the largest weighted F-measure under ⟨CVE + SF⟩ corpus (i.e., LOGISTIC REGRESSION with TF-IDF and oversampled training data) would score 0.54 with the simple average and 0.64 with the median, noticeably higher than 0.49 weighted score. We believe such inflated values do not accurately describe the overall model performance, motivating the use of a weighted aggregator. Still, we opted to closely inspect the model trends over the 22 validation rounds to avoid concluding using only a single aggregated value.

The Latent Semantic Analysis (LSA) employed in Section 4.3 allowed us to inspect the arrangement of the training and test instances at key validation rounds. We opted for this technique due to its suitability for textual data based on document-term matrices, which are known to generate a sparse feature space [31]. We were careful to fit the semantic space only on the training data to prevent it from being influenced by future data that was supposed to be unseen at that time. In other words, the test data were projected in the same semantic space previously fitted on the corresponding training set.

7 CONCLUSION

This paper presented a large-scale empirical evaluation of the effectiveness of *early* exploitability prediction models relying on the data available in a just-disclosed vulnerability, comparing 72 learning configurations, involving six traditional ML classifiers, four feature representation schemas, and three data balancing settings, as well as five pre-trained LLMs. All models were evaluated in the context of a *time-aware* validation setting representing a *realistic* scenario where the models are periodically re-trained and deployed. Additionally, we handled possible issues connected to an unrealistic and eager assignment of labels by employing a special data cleaning strategy.

The results showed that CVE descriptions alone suffice, but the addition of online discussions from SECURITY FOCUS further boosts the performance of any model. The best combination of feature representation and data balancing was with TF and SMOTE in the majority of the cases. The best classifier depends on the performance metric: the LOGISTIC REGRESSION achieved the best F-measure and MCC scores, the RANDOM FOREST maximized the precision, and the KNN had a quasi-perfect recall. Unfortunately, pre-trained LLMs did not achieve the expected performance, requiring further pre-training in the security domain. Nevertheless, all models fell victim to the same phenomenon, i.e., a noticeable drop at later validation rounds, likely due to the large imbalance in the test sets.

Future research directions include the experimentation of novel mechanisms to build a more reliable and sound ground truth—e.g., by combining multiple data sources—or alternative learning configurations to improve the *early* exploitability prediction model performance. We envision possible further developments to make exploitability prediction more powerful and useful, such as employing an *incremental* exploitability prediction system to guide the choice of which countermeasures to apply when a new vulnerability is published, e.g., helping to decide which vulnerability

must be addressed before than others. Such a system can also be integrated with an *impact estimation* module to provide a full overview of the risk connected to a newly found vulnerability. From a different perspective, we hypothesize that exploitability prediction modeling can be further improved by retrieving peculiar information from the software systems affected by just-disclosed vulnerabilities and using fine-grained text analysis tools that extract relevant elements from unstructured text, such as code snippets or stack traces, to have a more relevant feature space from which the models can better learn.

CREDITS

Emanuele Iannone: Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Giulia Sellitto:** Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Emanuele Iaccarino:** Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Filomena Ferrucci:** Supervision, Validation, Writing - Review & Editing. **Andrea De Lucia:** Supervision, Validation, Writing - Review & Editing. **Fabio Palomba:** Supervision, Validation, Writing - Review & Editing.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

DATA AVAILABILITY

The datasets built during the current study, plus the scripts used to analyze and generate the data, are available in the FIGSHARE repository: <https://figshare.com/s/165b39c7094c7831365e>.

ACKNOWLEDGMENTS

This work has been partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY). This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

REFERENCES

- [1] Luca Allodi and Fabio Massacci. 2014. Comparing Vulnerability Severity and Exploits Using Case-Control Studies. *ACM Trans. Inf. Syst. Secur.* 17, 1, Article 1 (aug 2014), 20 pages. <https://doi.org/10.1145/2630069>
- [2] Mohammed Almkaynizi, Eric Nunes, Krishna Dharaiya, Manoj Senguttuvan, Jana Shakarian, and Paulo Shakarian. 2017. Proactive identification of exploits in the wild through vulnerability mentions online. In *2017 International Conference on Cyber Conflict (CyCon U.S.)*, 82–88. <https://doi.org/10.1109/CYCONUS.2017.8167501>
- [3] Yasmin AlNoamany, Ahmed AlSum, Michele C. Weigle, and Michael L. Nelson. 2014. Who and what links to the Internet Archive. *International Journal on Digital Libraries* 14, 3 (01 Aug 2014), 101–115. <https://doi.org/10.1007/s00799-014-0111-5>
- [4] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. 2020. Automation of Vulnerability Classification from its Description using Machine Learning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, 1–7. <https://doi.org/10.1109/ISCC50000.2020.9219568>
- [5] Ashish Arora, Anand Nandkumar, and Rahul Telang. 2006. Does information security attack frequency increase with vulnerability disclosure? An empirical analysis. *Information Systems Frontiers* 8 (01 2006), 350–362. <https://doi.org/10.1007/s10796-006-9012-5>
- [6] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2020. Dos and Don'ts of Machine Learning in Computer Security. In *Proc. of USENIX Security Symposium*. <http://arxiv.org/abs/2010.09470>
- [7] Andrei Arusoae, Stefan Ciobăca, Vlad Craciun, Dragos Gavrilut, and Dorel Lucanu. 2017. A Comparison of Open-Source Static Analysis Tools for Vulnerability Detection in C/C++ Code. In *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 161–168. <https://doi.org/10.1109/SYNASC.2017.00035>

- [8] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. ACM Press.
- [9] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus A. F. Andersen, and Henrik Nielsen. 2000. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics* 16, 5 (05 2000), 412–424. <https://doi.org/10.1093/bioinformatics/16.5.412> arXiv:https://academic.oup.com/bioinformatics/article-pdf/16/5/412/48836094/bioinformatics_16_5_412.pdf
- [10] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Extracting Structural Information from Bug Reports. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories* (Leipzig, Germany) (*MSR '08*). Association for Computing Machinery, New York, NY, USA, 27–30. <https://doi.org/10.1145/1370750.1370757>
- [11] Nicolas Bettenburg, Emad Shihab, and Ahmed E. Hassan. 2009. An empirical study on the risks of using off-the-shelf techniques for processing mailing list data. In *2009 IEEE International Conference on Software Maintenance*. 539–542. <https://doi.org/10.1109/ICSM.2009.5306383>
- [12] Navneet Bhatt, Adarsh Anand, and V. S. S. Yadavalli. 2021. Exploitability prediction of software vulnerabilities. *Quality and Reliability Engineering International* 37, 2 (2021), 648–663. <https://doi.org/10.1002/qre.2754> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.2754>
- [13] Leyla Bilge and Tudor Dumitraş. 2012. Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) (*CCS '12*). Association for Computing Machinery, New York, NY, USA, 833–844. <https://doi.org/10.1145/2382196.2382284>
- [14] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2010. Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Washington, DC, USA) (*KDD '10*). Association for Computing Machinery, New York, NY, USA, 105–114. <https://doi.org/10.1145/1835804.1835821>
- [15] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
- [16] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. 1984. *Classification and regression trees Regression trees*. Chapman and Hall.
- [17] Benjamin L. Bullough, Anna K. Yanchenko, Christopher L. Smith, and Joseph R. Zipkin. 2017. Predicting Exploitation of Disclosed Software Vulnerabilities Using Open-Source Data. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics* (Scottsdale, Arizona, USA) (*IWSPA '17*). Association for Computing Machinery, New York, NY, USA, 45–53. <https://doi.org/10.1145/3041008.3041009>
- [18] Gerardo Canfora, Andrea Di Sorbo, Sara Forootani, Antonio Pirozzi, and Corrado Aaron Visaggio. 2020. Investigating the vulnerability fixing process in OSS projects: Peculiarities and challenges. *Computers & Security* 99 (2020), 102067. <https://doi.org/10.1016/j.cose.2020.102067>
- [19] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. 2007. Efficiency of Vulnerability Disclosure Mechanisms to Disseminate Vulnerability Knowledge. *IEEE Transactions on Software Engineering* 33, 3 (2007), 171–185. <https://doi.org/10.1109/TSE.2007.26>
- [20] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the Quality of the Steps to Reproduce in Bug Reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) (*ESEC/FSE 2019*). Association for Computing Machinery, New York, NY, USA, 86–96. <https://doi.org/10.1145/3338906.3338947>
- [21] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority over-Sampling Technique. *J. Artif. Int. Res.* 16, 1 (jun 2002), 321–357.
- [22] Hai Leong Chieu and Hwee Tou Ng. 2002. A maximum entropy approach to information extraction from semi-structured and free text. *Aaai/iaai 2002* (2002), 786–791.
- [23] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. arXiv:1911.02116 [cs.CL]
- [24] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
- [25] Roland Croft, M. Ali Babar, and Li Li. 2022. An Investigation into Inconsistency of Software Vulnerability Severity across Data Sources. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 338–348. <https://doi.org/10.1109/SANER53432.2022.00050>
- [26] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2013. Applying a smoothing filter to improve IR-based traceability recovery processes: An empirical investigation. *Information and Software Technology* 55, 4 (2013), 741–754. <https://doi.org/10.1016/j.infsof.2012.08.002>
- [27] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

- [28] Peter Devine, Yun Sing Koh, and Kelly Blincoe. 2021. Evaluating Unsupervised Text Embeddings on Software User Feedback. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. 87–95. <https://doi.org/10.1109/REW53955.2021.00020>
- [29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. <https://arxiv.org/abs/1810.04805>
- [30] Nesara Dissanayake, Asangi Jayatilaka, Mansoor Zahedi, and M. Ali Babar. 2022. Software security patch management - A systematic literature review of challenges, approaches, tools and practices. *Information and Software Technology* 144 (2022), 106771. <https://doi.org/10.1016/j.infsof.2021.106771>
- [31] Susan T. Dumais. 2004. Latent semantic analysis. *Annual Review of Information Science and Technology* 38, 1 (2004), 188–230. <https://doi.org/10.1002/aris.1440380105> arXiv:<https://arxiv.org/abs/2004.04001>
- [32] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference (Vancouver, BC, Canada) (IMC '14)*. Association for Computing Machinery, New York, NY, USA, 475–488. <https://doi.org/10.1145/2663716.2663755>
- [33] Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1, 3 (01 Sep 1936), 211–218. <https://doi.org/10.1007/BF02288367>
- [34] Michel Edkrantz and Alan Said. 2015. Predicting Cyber Vulnerability Exploits with Machine Learning. In *SCAI*. 48–57. <https://doi.org/10.3233/978-1-61499-589-0-48>
- [35] Clément Elbaz, Louis Rilling, and Christine Morin. 2020. Fighting N-Day Vulnerabilities with Automated CVSS Vector Prediction at Disclosure. In *Proceedings of the 15th International Conference on Availability, Reliability and Security (Virtual Event, Ireland) (ARES '20)*. Association for Computing Machinery, New York, NY, USA, Article 26, 10 pages. <https://doi.org/10.1145/3407023.3407038>
- [36] Michael Felderer, Matthias Büchler, Martin Johns, Achim D. Brucker, Ruth Breu, and Alexander Pretschner. 2016. Chapter One - Security Testing: A Survey. *Advances in Computers*, Vol. 101. Elsevier, 1–51. <https://doi.org/10.1016/b.s.adcom.2015.11.003>
- [37] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. arXiv:2002.08155 [cs.CL]
- [38] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. 2006. Large-Scale Vulnerability Analysis. In *Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense (Pisa, Italy) (LSAD '06)*. Association for Computing Machinery, New York, NY, USA, 131–138. <https://doi.org/10.1145/1162666.1162671>
- [39] Stefan Frei, Dominik Schatzmann, Bernhard Plattner, and Brian Trammell. 2010. *Modeling the Security Ecosystem - The Dynamics of (In)Security*. 79–106. https://doi.org/10.1007/978-1-4419-6967-5_6
- [40] Milton Friedman. 1940. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics* 11, 1 (1940), 86–92.
- [41] Christian Fruhwirth and Tomi Mannisto. 2009. Improving CVSS-based vulnerability prioritization and response with context information. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 535–544. <https://doi.org/10.1109/ESEM.2009.5314230>
- [42] Aayush Garg, Renzo Degiovanni, Matthieu Jimenez, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2022. Learning from what we know: How to perform vulnerability prediction using noisy historical data. *Empirical Software Engineering* 27, 7 (20 Sep 2022), 169. <https://doi.org/10.1007/s10664-022-10197-4>
- [43] Luiz Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. 2023. BERT- and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study. *Information and Software Technology* 160 (2023), 107217. <https://doi.org/10.1016/j.infsof.2023.107217>
- [44] Yang Guo. 2023. A review of Machine Learning-based zero-day attack detection: Challenges and future directions. *Computer Communications* 198 (2023), 175–185. <https://doi.org/10.1016/j.comcom.2022.11.001>
- [45] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 125–136. <https://doi.org/10.1109/ICSME.2017.52>
- [46] Hannes Holm, Mathias Ekstedt, and Dennis Andersson. 2012. Empirical Analysis of System-Level Vulnerability Metrics through Actual Attacks. *IEEE Transactions on Dependable and Secure Computing* 9, 6 (2012), 825–837. <https://doi.org/10.1109/TDSC.2012.66>
- [47] Kaifeng Huang, Bihuan Chen, Linghao Pan, Shuai Wu, and Xin Peng. 2021. REPFINDER: Finding Replacements for Missing APIs in Library Update. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 266–278.

- [48] Emanuele Iannone, Roberta Guadagni, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2023. The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study. *IEEE Transactions on Software Engineering* 49, 1 (2023), 44–63. <https://doi.org/10.1109/TSE.2022.3140868>
- [49] Emanuele Iannone, Giulia Sellitto, Emanuele Iaccarino, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2023. Early and Realistic Exploitability Prediction of Just-Disclosed Software Vulnerabilities: How Reliable Can It Be? – Online Appendix. <https://figshare.com/s/165b39c7094c7831365e>.
- [50] Jay Jacobs, Sasha Romanosky, Idris Adjerid, and Wade Baker. 2020. Improving vulnerability remediation through better exploit prediction. *Journal of Cybersecurity* 6, 1 (09 2020). <https://doi.org/10.1093/cybsec/tyaa015> arXiv:<https://academic.oup.com/cybersecurity/article-pdf/6/1/tyaa015/33746021/tyaa015.pdf> tyaa015.
- [51] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Idris Adjerid, and Michael Roytman. 2021. Exploit Prediction Scoring System (EPSS). *Digital Threats* 2, 3, Article 20 (jul 2021), 17 pages. <https://doi.org/10.1145/3436242>
- [52] J. Jacobs, S. Romanosky, O. Suci, B. Edwards, and A. Sarabi. 2023. Enhancing Vulnerability Prioritization: Data-Driven Exploit Predictions with Community-Driven Insights. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*. IEEE Computer Society, Los Alamitos, CA, USA, 194–206. <https://doi.org/10.1109/EuroSPW59978.2023.00027>
- [53] Zhiyuan Jiang, Shuitao Gan, Adrian Herrera, Flavio Toffalini, Lucio Romerio, Chaojing Tang, Manuel Egele, Chao Zhang, and Mathias Payer. 2022. Evocatio: Conjuring Bug Capabilities from a Single PoC. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (Los Angeles, CA, USA) (CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1599–1613. <https://doi.org/10.1145/3548606.3560575>
- [54] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. 2019. The Importance of Accounting for Real-World Labelling When Predicting Software Vulnerabilities. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Tallinn, Estonia) (ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 695–705. <https://doi.org/10.1145/3338906.3338941>
- [55] Atefeh Khazaei, Mohammad Ghasemzadeh, and Vali Derhami. 2016. An automatic method for CVSS score prediction using vulnerabilities description. *Journal of Intelligent & Fuzzy Systems* 30 (2016), 89–96. <https://doi.org/10.3233/IFS-151733> 1.
- [56] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2018. Do developers update their library dependencies? *Empirical Software Engineering* 23, 1 (01 Feb 2018), 384–417. <https://doi.org/10.1007/s10664-017-9521-5>
- [57] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. arXiv:1909.11942 [cs.CL]
- [58] William Landi. 1992. Undecidability of Static Analysis. *ACM Lett. Program. Lang. Syst.* 1, 4 (Dec. 1992), 323–337. <https://doi.org/10.1145/161494.161501>
- [59] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. PMLR, 1188–1196.
- [60] Triet Huynh Minh Le, David Hin, Roland Croft, and M. Ali Babar. 2021. DeepCVA: Automated Commit-Level Vulnerability Assessment with Deep Multi-Task Learning. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (Melbourne, Australia) (ASE '21)*. IEEE Press, 717–729. <https://doi.org/10.1109/ASE51524.2021.9678622>
- [61] Xueying Li, Peng Liang, and Zengyang Li. 2020. Automatic Identification of Decisions from the Hibernate Developer Mailing List. In *Proceedings of the Evaluation and Assessment in Software Engineering (Trondheim, Norway) (EASE '20)*. Association for Computing Machinery, New York, NY, USA, 51–60. <https://doi.org/10.1145/3383219.3383225>
- [62] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. 2018. Fuzzing: State of the Art. *IEEE Transactions on Reliability* 67, 3 (2018), 1199–1218. <https://doi.org/10.1109/TR.2018.2834476>
- [63] Bohan Liu, He Zhang, Lanxin Yang, Liming Dong, Haifeng Shen, and Kaiwen Song. 2020. An Experimental Evaluation of Imbalanced Learning and Time-Series Validation in the Context of CI/CD Prediction. In *Proceedings of the Evaluation and Assessment in Software Engineering (Trondheim, Norway) (EASE '20)*. Association for Computing Machinery, New York, NY, USA, 21–30. <https://doi.org/10.1145/3383219.3383222>
- [64] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]
- [65] Francesco Lomio, Emanuele Iannone, Andrea De Lucia, Fabio Palomba, and Valentina Lenarduzzi. 2022. Just-in-time software vulnerability detection: Are we there yet? *Journal of Systems and Software* 188 (2022), 111283. <https://doi.org/10.1016/j.jss.2022.111283>
- [66] Jinghui Lyu, Yude Bai, Zhenchang Xing, Xiaohong Li, and Weimin Ge. 2021. A Character-Level Convolutional Neural Network for Predicting Exploitability of Vulnerability. In *2021 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. 119–126. <https://doi.org/10.1109/TASE52547.2021.00014>

- [67] Fabio Massacci and Viet Hung Nguyen. 2010. Which is the Right Source for Vulnerability Studies? An Empirical Analysis on Mozilla Firefox. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics (Bolzano, Italy) (MetriSec '10)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. <https://doi.org/10.1145/1853919.1853925>
- [68] Brian W. Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405, 2 (1975), 442–451.
- [69] G. McGraw. 2006. *Software Security: Building Security in*. Addison-Wesley.
- [70] Scott Menard. 2002. *Applied logistic regression analysis*. Vol. 106. Sage.
- [71] Tomás Mikolov, Quoc V. Le, and Ilya Sutskever. 2013. Exploiting Similarities among Languages for Machine Translation. *CoRR* abs/1309.4168 (2013). arXiv:1309.4168 <http://arxiv.org/abs/1309.4168>
- [72] Patrick Morrison, Kim Herzig, Brendan Murphy, and Laurie Williams. 2015. Challenges with Applying Vulnerability Prediction Models. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security (Urbana, Illinois) (HotSoS '15)*. Association for Computing Machinery, New York, NY, USA, Article 4, 9 pages. <https://doi.org/10.1145/2746194.2746198>
- [73] Peter Bjorn Nemenyi. 1963. *Distribution-free multiple comparisons*. Princeton University.
- [74] Fabiano Pecorelli, Dario Di Nucci, Coen De Roover, and Andrea De Lucia. 2020. A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. *Journal of Systems and Software* 169 (2020), 110693. <https://doi.org/10.1016/j.jss.2020.110693>
- [75] Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2021. JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction. 369–379. <https://doi.org/10.1109/MSR52588.2021.00049>
- [76] Martin F. Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (01 Jan 1980), 130–137. <https://doi.org/10.1108/eb046814>
- [77] David M. W. Powers. 2011. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63.
- [78] Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E. Hassan. 2018. Revisiting the Performance Evaluation of Automated Approaches for the Retrieval of Duplicate Issue Reports. *IEEE Transactions on Software Engineering* 44, 12 (2018), 1245–1268. <https://doi.org/10.1109/TSE.2017.2755005>
- [79] John Ratcliff and David Metzener. 1988. Pattern Matching: the Gestalt Approach. <https://www.drdoobs.com/database/pattern-matching-the-gestalt-approach/184407970?pgno=5>. Accessed: 2022-03-125.
- [80] Alexander Reinthal, Eleftherios Lef Filippakis, and Magnus Almgren. 2018. Data Modelling for Predicting Exploits. In *Secure IT Systems*, Nils Gruschka (Ed.). Springer International Publishing, Cham, 336–351.
- [81] Stephan Rénatus, Corrie Bartelheimer, and Jörn Eichler. 2015. Improving prioritization of software weaknesses using security models with AVUS. In *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 259–264.
- [82] Thomas Reps. 2000. Undecidability of Context-Sensitive Data-Dependence Analysis. *ACM Trans. Program. Lang. Syst.* 22, 1 (Jan. 2000), 162–186. <https://doi.org/10.1145/345099.345137>
- [83] Irina Rish. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3. 41–46.
- [84] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. *Automatic Keyword Extraction from Individual Documents*. John Wiley & Sons, Ltd, Chapter 1, 1–20. <https://doi.org/10.1002/9780470689646.ch1> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470689646.ch1>
- [85] Carl Sabottke, Octavian Suciu, and Tudor Dumitras. 2015. Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits. In *Proceedings of the 24th USENIX Conference on Security Symposium (Washington, D.C.) (SEC'15)*. USENIX Association, USA, 1041–1056.
- [86] Pasquale Salza, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Filomena Ferrucci. 2020. Third-party libraries in mobile apps. *Empirical Software Engineering* 25, 3 (2020), 2341–2377.
- [87] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 [cs.CL]
- [88] Arthur D. Sawadogo, Quentin Guimard, Tegawendé F. Bissyandé, Abdoul Kader Kaboré, Jacques Klein, and Naouel Moha. 2021. Early Detection of Security-Relevant Bug Reports using Machine Learning: How Far Are We? *CoRR* abs/2112.10123 (2021). arXiv:2112.10123 <https://arxiv.org/abs/2112.10123>
- [89] Simone Scalabrino, Mario Linares-Vásquez, Rocco Oliveto, and Denys Poshyvanyk. 2018. A comprehensive model for code readability. *Journal of Software: Evolution and Process* 30, 6 (2018), e1958. <https://doi.org/10.1002/smr.1958> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1958> e1958 smr.1958.
- [90] Adrian Schroter, Adrian Schröter, Nicolas Bettenburg, and Rahul Premraj. 2010. Do stack traces help developers fix bugs?. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 118–121. <https://doi.org/10.1109/MSR.2010.5463280>

- [91] C. Silva and B. Ribeiro. 2003. The importance of stop word removal on recall values in text categorization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, Vol. 3. 1661–1666 vol.3. <https://doi.org/10.1109/IJCNN.2003.1223656>
- [92] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. 2020. The significance of bug report elements. *Empirical Software Engineering* 25, 6 (01 Nov 2020), 5255–5294. <https://doi.org/10.1007/s10664-020-09882-z>
- [93] Octavian Suci, Connor Nelson, Zhuoer Lyu, Tiffany Bao, and Tudor Dumitras. 2022. Expected Exploitability: Predicting the Development of Functional Vulnerability Exploits. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 377–394. <https://www.usenix.org/conference/usenixsecurity22/presentation/suci>
- [94] Christopher Theisen and Laurie Williams. 2020. Better together: Comparing vulnerability prediction models. *Information and Software Technology* 119 (2020), 106204. <https://doi.org/10.1016/j.infsof.2019.106204>
- [95] Victor R. V. R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach. *Encyclopedia of Software Engineering* (1994).
- [96] John Viega, J.T. Bloch, Y. Kohno, and Gary McGraw. 2000. ITS4: a static vulnerability scanner for C and C++ code. In *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*. 257–267. <https://doi.org/10.1109/ACSAC.2000.898880>
- [97] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (01 Jul 2016), 964–994. <https://doi.org/10.1007/s10618-015-0448-4>
- [98] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144 [cs.CL]
- [99] Jingxiu Yao and Martin Shepperd. 2020. Assessing Software Defection Prediction Performance: Why Using the Matthews Correlation Coefficient Matters. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (Trondheim, Norway) (EASE '20)*. Association for Computing Machinery, New York, NY, USA, 120–129. <https://doi.org/10.1145/3383219.3383232>
- [100] Jiao Yin, MingJian Tang, Jinli Cao, and Hua Wang. 2020. Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description. *Knowledge-Based Systems* 210 (2020), 106529. <https://doi.org/10.1016/j.knosys.2020.106529>
- [101] Jiao Yin, MingJian Tang, Jinli Cao, Hua Wang, and Mingshan You. 2022. A real-time dynamic concept adaptive learning algorithm for exploitability prediction. *Neurocomputing* 472 (2022), 252–265. <https://doi.org/10.1016/j.neucom.2021.01.144>
- [102] Jianping Zhang and Inderjeet Mani. 2003. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*.
- [103] Yun Zhang, David Lo, Xin Xia, Bowen Xu, Jianling Sun, and Shanping Li. 2015. Combining Software Metrics and Text Features for Vulnerable File Prediction. In *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*. 40–49. <https://doi.org/10.1109/ICECCS.2015.15>
- [104] Zhongheng Zhang. 2016. Introduction to machine learning: k-nearest neighbors. *Annals of translational medicine* 4, 11 (2016).
- [105] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. 2021. Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 705–716. <https://doi.org/10.1109/ASE51524.2021.9678720>
- [106] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. 2016. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process* 28, 3 (2016), 150–176. <https://doi.org/10.1002/smr.1770> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1770>