

On Predicting Vulnerability Severity Using In-Context Learning: An Industrial Case Study

Daniel Rodriguez-Cardenas 
William & Mary
 Williamsburg, VA, USA
 dhrodriguezcar@wm.edu

David Nader Palacio 
Microsoft
 Seattle, WA, USA
 dnader@microsoft.com

Anna Schmedding
William & Mary
 Williamsburg, VA, USA
 akschmedding@wm.edu

Yiyang Lu
William & Mary
 Williamsburg, VA, USA
 ylu21@wm.edu

Aadil Mallick
William & Mary
 Williamsburg, VA, USA
 amallick@wm.edu

Bill Hudson
Cisco Systems
 San Jose, CA, USA
 bhudson@cisco.com

Chris Gourley
Cisco Systems
 San Jose, CA, USA
 chgourle@cisco.com

Michael Roytman
Cisco Systems
 San Jose, CA, USA
 roytman@cisco.com

Chris Shenefiel
William & Mary
 Williamsburg, VA, USA
 cashenefiel@wm.edu

Evgenia Smirni
William & Mary
 Williamsburg, VA, USA
 esmirni@wm.edu

Denys Poshyvanyk 
William & Mary
 Williamsburg, VA, USA
 denys@cs.wm.edu

Abstract—Modern software systems require earlier and more scalable vulnerability severity assessment to reduce exposure to high-impact security flaws. Security analysts typically assign CVSS scores, but this manual triage does not scale with the growth of disclosed vulnerabilities and often depends on cloud LLM services that raise confidentiality concerns. This paper presents an industrial case study on predicting CVSS v3.1 scores directly from vulnerable C/C++ snippets using *in-context learning* with locally deployable, *open-source* LLMs. We compare *Proprietary* data with the *Big-Vul* dataset, showing sufficiently aligned CVSS distributions to justify *Big-Vul* as a proxy for industrial data when constructing prompt-based testbeds. We then vary in-context configurations and model parameters, evaluating *CodeLlama2-7B*, *CodeLlama2-13B*, *Mistral-7B*, *gpt-oss*, and *GPT4o-mini* using mean squared error (MSE) and feasibility metrics. Our results show that medium-sized open-source code models, particularly *CodeLlama2-7B*, can approximate the best cloud performance for CVSS regression when guided by lightweight, output-constraining prompts, offering a practical, privacy-preserving building block for severity triage in industrial settings.

Index Terms—Security, Large Language Models, Vulnerability, Prediction, Severity

I. INTRODUCTION

Detecting risky flaws in software has become an instrumental practice for large companies and start-ups alike, as modern systems rapidly evolve. Attackers exploit these vulnerabilities, leading to more complex issues such as data leakage, service interruptions, and increased operational costs [20], [21]. In

industry, vulnerability management pipelines typically involve identifying, prioritizing, and repairing vulnerabilities in various artifacts such as code, configuration files, documentation, and architectural diagrams. Prioritization is based on severity estimation through standardized risk metrics, among which the Common Vulnerability Scoring System (CVSS) has emerged as the de facto severity score -, widely adopted by major vendors and the National Vulnerability Database (NVD) [8], [11], [12].

Performing accurate CVSS score prediction [11], [12] is both critical and challenging. Studies show that only a small fraction of exploited vulnerabilities are initially scored and prioritized correctly, and that roughly half of real-world exploits occur before CVSS scores are published, creating a dangerous window in which organizations operate with incomplete severity information. At the same time, alternative risk scores, such as Cisco’s Kenna Risk [47] and the Exploit Prediction Scoring System (EPSS) [11] build directly on CVSS to quantify exploitability and business impact, confirming the central role of severity in downstream risk models. However, assigning CVSS scores remains a manual and time-consuming task - that does not scale with the growing volume of disclosed vulnerabilities and code changes.

Recent advances in Large Language Models (LLMs) suggest that AI systems can assist analysts by classifying vulnerabilities, explaining flaws [13], [25], [50], and even suggesting code fixes. Cloud - hosted assistants, such as Chat-

GPT [14], Gemini [1], and Claude [3] have demonstrated strong performance on security-related tasks, but their use in practice is constrained by confidentiality, compliance, and data residency requirements that prevent organizations from uploading proprietary source code and incident data to third-provider [12], [50]. These constraints have sparked interest in *open-source*, locally deployable LLMs that can run on customer infrastructure and operate on internal codebases without exposing sensitive artifacts. Although several works have explored machine learning and deep learning models for CVSS prediction using textual descriptions, little is known about how code-centric open-source LLMs perform in severity regression when guided by in-context learning.

This paper investigates whether *in-context learning* with *open-source* code LLMs can support earlier, privacy-preserving CVSS v3.1 [11] severity prediction directly from vulnerable C/C++ snippets. We first compare a *Proprietary* dataset with the public *Big-Vul* corpus and show that their CVSS distributions overlap in the 4–8 range and exhibit similar correlations with adjusted scores, supporting the use of *Big-Vul* as a proxy for industrial data when constructing prompt-based testbeds. Building on this foundation, we design an in-context learning pipeline that assembles prompts from vulnerable code, CVSS scores, and descriptions, and systematically vary (i) context configuration (C_1 – C_3), (ii) sequence length, (iii) temperature, and (iv) number of shots. We evaluate multiple *open-source* (i.e., *CodeLlama2-7B*, *CodeLlama2-13B*, *Mistral-7B*, *gpt-oss*) and the cloud model *GPT4o-mini* as a reference using mean squared error (MSE), *Non-Feasible* output rate and *Out-Of-Range* predictions as our primary metrics. Our results show that *CodeLlama2-7B* is the most reliable open-source option, if a vulnerability inspector wants a local and small option, achieving *MSE* around 7 with three-shot prompts and virtually *Non-Feasible* outputs, while *GPT4o-mini* under a constrained context (C_3) attains substantially lower *MSE* at the cost of requiring cloud deployment.

In this work, we restrict ourselves to few-shot prompting, focusing on simple *in-context* configurations rather than more sophisticated reasoning pipelines. Although this keeps our setup aligned with practical, low-latency deployment constraints, it also leaves open the opportunity to explore richer prompting strategies—such as chain-of-thought, self-reflection, or “thinking” models with built-in reasoning capabilities—as an important direction for future work to further improve the reliability and interpretability of severity predictions.

In summary, this work makes the following contributions:

- We present an exploratory analysis of *Proprietary* vulnerability data and the *Big-Vul* dataset, demonstrating overlapping CVSS distributions and aligned correlations with Kenna-adjusted scores, which justifies using *Big-Vul* as an open proxy for industrial severity modeling.
- We design and evaluate a family of *in-context configurations* (C_1 – C_3) for CVSS regression, showing that adding a small number of examples and restricting the

output format are the key to reducing MSE, non-feasible responses, and scores out-of-range between models.

- We empirically compare *open-source* and cloud LLMs for CVSS prediction and find that medium-sized, locally deployable code models—particularly *CodeLlama2-7B*—offer a practical trade-off between accuracy and privacy, approximating cloud-model performance while keeping sensitive artifacts on-premise.
- Finally, we built a *Big-Vul* based testbed and shared our complete empirical evaluation artifacts to facilitate reproducibility [40].

II. BACKGROUND

The Common Vulnerability Scoring System (CVSS) is a standardized scoring method to assess the severity of a computer system security vulnerability. CVSS v3 comprises three base metric groups [12]. The first group is based on exploitability metrics that include the attack vector, the complexity of the attack, the required privileges, and user interaction. The second group is the impact metric, which includes confidentiality, integrity, and availability. The third group is the scope, designed to capture the impact of the vulnerable component on its linked components. CVSS is the main score industry vendors (e.g., Microsoft, Google, Apple, Linux) use to report and prioritize vulnerabilities. We focus on predicting the CVSS score due to its widespread use in industry and association with public datasets. Alternatives such as stakeholder-specific vulnerability categorization (SSVC) and vulnerability priority rating (VPR) incorporate additional factors for severity estimation. SSVC uses a decision tree model for prioritization [43], while VPR adds tracking factors and vulnerability history to CVSS V3 [45].

The National Vulnerability Database (NVD) is a comprehensive cybersecurity vulnerability database maintained by the National Institute of Standards and Technology (NIST), the U.S. Department of Commerce agency. NVD serves as the U.S. government repository of standards-based vulnerability management data and acts as the major publisher of the CVSS scores for almost all known vulnerabilities.

The CVSS score is focused on the severity of the vulnerability, new metrics have been proposed to bring more information, for example, the Exploit Prediction Scoring System (EPSS) introduced by Jacobs et al. [22], aims to predict the probability that a vulnerability will be exploited in the wild. It provides a probability between 0 and 1, indicating the chance of a vulnerability being exploited within the next 30 days. EPSS, in its first version, uses a logistic regression model trained on 16 features to predict the probability of exploitation within the first year of a vulnerability’s disclosure. For the second and third versions of EPSS the logistic regression model switched to a centralized architecture using XGBoost [7] improving predictive performance.

The Common Vulnerability and Exposures (CVE), found in 1999, is a dictionary of vulnerabilities that have been identified in various code bases [30]. Each vulnerability is associated with a unique identifier called CVE ID. The CVE program

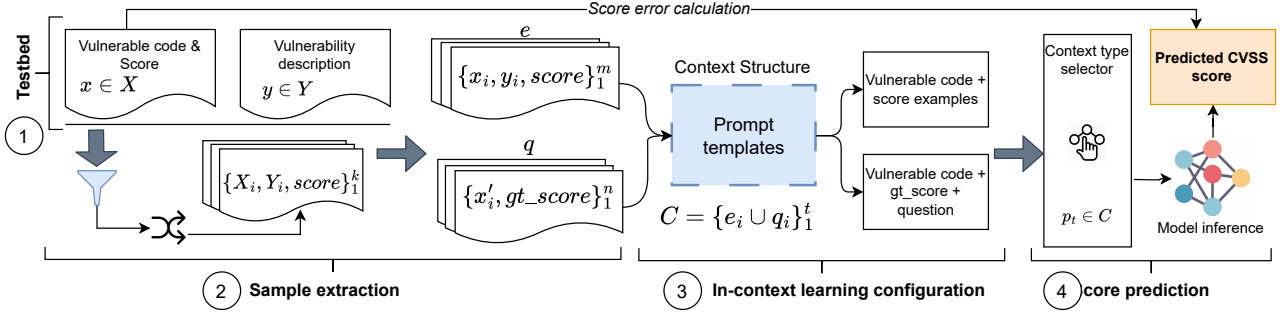


Fig. 1: In-Context Learning Solution as a Severity Score Prediction Pipeline

is maintained by MITRE corporation and sponsored by the U.S. Department of Homeland Security(DHS) and the Cybersecurity and Infrastructure Security Agency(CISA). Efforts have proposed using severity to calculate a vulnerability's exploitability using heuristics and context aggregation (e.g., social media) [6], [10], [42], [51]. Cisco's Kenna risk score evaluates vulnerabilities based on organizational risk, using machine learning for dynamic scoring. The Exploit Prediction Scoring System (EPSS) predicts a vulnerability's exploitation likelihood within 30 days. Both Kenna and EPSS use CVSS scores to predict exploitability. This report evaluates LLMs with CVSS, noting their time-series forecasting limits [44]. Other LLMs categorize vulnerabilities and fix code [16], [17], [26], [41].

Vulnerability Severity Life Cycle. *Vulnerability management* focuses on detecting, triaging, and enumerating mitigation action. Therefore, security analysts take into account several dates in the vulnerability life cycle. 1) A new software feature is added or modified, 2) the new feature is deployed, 3) a vulnerability is identified, and then, depending on the severity, a fix is required sooner or later. As a consequence, 4.1) a new fix is released and/or 4.2) an attacker exploits the vulnerability. Recent reports indicate a trend in the reduction time from vulnerability detection and attacker exploit, which means that early detection and assisting security analysis is crucial, especially in safety-critical software.

In-Context Learning for Vulnerability Severity. Due to the expensive pre-training and fine-tuning process for adapting LLMs to a specific task, in-context learning emerges as a strategy for guiding the model to find an accurate answer [15], [29]. in-context learning consists of designing a set of interactions using input prompt templates [27]. In the realm of vulnerability prediction, in-context Learning emerges as a pivotal strategy to bolster model efficacy and adaptability. By embracing in-context learning, LLMs grasp nuanced contextual cues inherent in vulnerability data, thus fortifying their predictive capabilities and adaptability to varying threat landscapes [14], [28]. Our approach aims to apply in-context learning to *open-source* models that can be executed on the customer infrastructure, as previous studies applied to cloud solutions.

III. APPROACH

Problem Delimitation. Any change or update in software systems produces a new feature that must be analyzed using severity score systems (i.e., CVSS) to assess the risks of releasing such a feature and preventing security threats (e.g., unauthorized access, data leakage, and system shutdown). However, predicting the severity of the vulnerability is challenging because security analysts must assess *the impact of a vulnerability within a limited time frame*. In addition, analysis becomes crucial and complex as the number of features and vulnerabilities increases. Therefore, a solution is to help the security analyst with tools that help to assess the severity impact and intervene in the vulnerability life cycle in earlier stages (i.e., code development when a new feature is added or modified).

This paper proposes the analysis of *open-source* models to assist analysts in the prioritization task by predicting the severity score.*open-source* models are particularly interesting for industry, as they can run locally, allowing software companies to maintain control over the environment while avoiding sending sensitive data to third-party AI cloud providers.

Several studies mainly focused on vulnerability classification (i.e., whether the code snippet is vulnerable or not) [13], [25], [52]. In our approach, the prediction of the severity score is considered as a regression problem instead of a classification problem (e.g., none [0], low [0.1-3.9], medium [4-6.9], high [7-8.9] and critical [9-10]). Predicting the severity score as a numeric value rather than a categorical label enables the security analyst to quantify severity and provides better insights into the model's precision and calibration—the difference between the predicted and ground-truth distribution—on vulnerability impact [18]. Predicting just the category level conceals the severity of a given vulnerability. For example, consider that an analyst can only predict severity categories. A predicted category severity of *medium* was obtained. The severity of the regression was actually a hidden/unknown value of 6.9. In this scenario, the security analyst failed to determine whether the severity value was close to the low or high categories, reducing the precision of the prediction. This precision is crucial for prompt remediation of the vulnerability and for reasoning about the issue. We hypothesize that this model calibration benefits from our in-context learning solution since severity scores are directly predicted.

In-Context Learning Solution. We aim to help security analysts triage software features according to the severity of the vulnerability predicted by in-context learning configurations. We require a set of *open-source* models to implement the solution.

Step ①: Testbed. To evaluate the performance of these models in predicting vulnerability severity, we first collected a set of vulnerable snippets $x \in X$ with their corresponding scores (i.e., CVSS, EPSS, Kenna Risk) and description $y \in Y$ (see Fig. 1). These snippets were collected from proprietary and open source repositories (refer to *Big-Vul* in Sec. VI-A). Our experience report only presents statistical analysis with open source snippets due to privacy agreements.

Step ②: Sample extraction The second step comprises sample extraction by filtering the code with more than 50 characters and up to 100 characters (*short* size sequence), and code with more than 100 characters and up to 300 characters (*medium* size sequence) with a valid description and CVSS score. Including a snippet of more than 50 characters ensures it includes at least the method signature and sufficient context. At the end of this step, we obtain the two subsets of triples $\{x_i, y_i, score\}$. Each subset is then used to create both a test dataset $M = \{x_i, y_i, score\}$ and a ground-truth dataset $N = \{x_i, y_i, gt_score\}$. We created the prompt *examples* for the in-context learning setup, using the M dataset, and the *questions* using the N dataset (Fig. 2). We always hide the ground-truth score (gt_score) from N to the model. A question will never be used as an example (Sec. VI-A). *Step ③: In-context Learning Configuration.* The third step presents the testbed generation using a set of prompt templates following a context-type structure (Fig. 2). A context-type structure concatenates the *example* and the *question*. We propose two context types (Sec. V). These context types require the vulnerable examples m with their current scores and the questions n to complete each prompt, for example e and each prompt for question q , respectively. For example, a question prompt template completes the sentence with the vulnerable code x'_i and question q .

Step ④: Prediction of scores The fourth step comprises the *context type selector* and the number of examples of shots $l \subset m_p \mid p \in C_t$ to be used. Therefore, the complete context interaction contains the l shot examples of prompt $p \in C_t$ type t with their vulnerable code x_i and the associated score, and the vulnerable code x'_i . We conceal the ground truth score gt_score of x'_i from the model. For the context type C_2 , we aggregate the code description in a context y_i . The experiments are executed on each LLM then we capture the predicted score value with a regex expression.

IV. METHODOLOGY

This section outlines the methodology employed to validate our interpretability technique, we conducted two case studies on xxx popular architectures to explore the following RQs:

V. RESEARCH QUESTIONS

In our experience report, we are interested in evaluating the datasets used in the industry and open datasets for vulnerability, the effectiveness of the in-context learning configuration, and the evaluation of *open-source* LLMs.

RQ₁ *What type of correlations and distributions are there in the industry dataset under study?* We aim to validate the similarity between the industry datasets used and the open-source datasets used to evaluate LLMs for vulnerability detection. The distribution shape allows us to understand the boundaries and limitations for building the in-context approach when using *open-source* models. In addition, because of limitations in reporting *Proprietary* datasets, distribution similarity serves as a proxy assess whether our evaluation generalizes equally to both industry and open-source settings.

RQ₂ *Which is the most effective in-context learning configuration to predict vulnerability severity scores?* We want to detect the most effective *in-context configuration* that enables LLMs to improve prediction performance. Fig. 2 illustrates two context templates we fed to our LLMs. The first context type C_1 is structured using two prompts. The first prompt contains an example of vulnerable code x along with its vulnerability severity $\langle score \rangle$, while the second prompt introduces vulnerable code x' asking for severity score. Notice that C_1 formulates the examples and questions about the vulnerability severity score without additional information (e.g., CVSS concept, ranges of scores, and description of vulnerability). The second context type C_2 comprises three prompts. The first prompt introduces the models' role, which is a code vulnerable inspector, and the range of the CVSS. This role aggregates more context to the prompt when no examples are available (i.e., zero-shot experiments). The second prompt concatenates the vulnerability description y , the vulnerable code example x , and the associated ground truth $\langle score \rangle$. Finally, the third prompt formulates a question specifically asking for the CVSS score of the corresponding vulnerable code x' .

RQ₃ *How accurate are open-source locally deployed LLMs at predicting vulnerability severity scores?* We are interested in evaluating the severity prediction accuracy as a regression problem. Predicting a score value is more informative and precise than classifying the vulnerability by severity ranges, since the score value represents specific submetrics (Sec. II) that provide complementary information for the security analyst. We aim to support security analysts in selecting the correct *open-source* model by monitoring the *mean square error*.

VI. CASE STUDY DESIGN

In this section, we describe the experimental design of our *in-context learning* solution to evaluate *open-source* LLMs to predict severity scores.

A. Setup

Testbeds. To build our *in-context learning configuration*, we extracted vulnerable code, scores, and description triplets

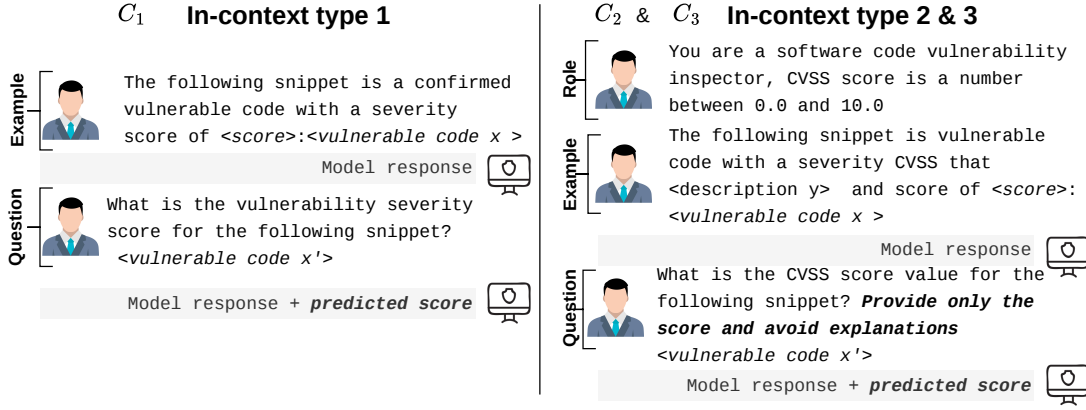


Fig. 2: In-Context Learning Configuration. C_3 aggregates a specific output format to answer the question compared to C_2 .

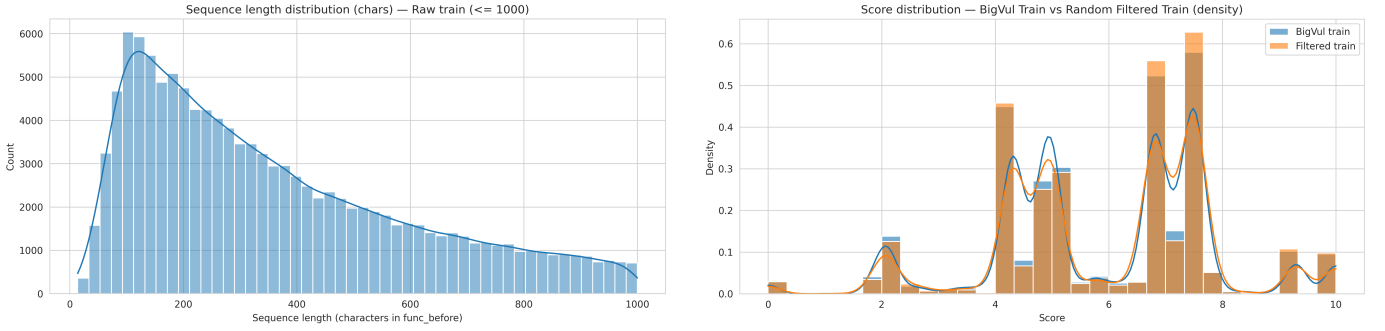


Fig. 3: Big-Vul dataset: on the left, the sequence size frequency; on the right, the CVSS score distribution for raw data and filtered dataset.

from *Big-Vul* [9]. *Big-Vul* is a dataset with $\approx 150k$ data points. We filter *Big-Vul* following the second step of our approach (Sec. III). *Big-Vul* already includes a curated dataset with the vulnerable code, description, and score. We compare *Big-Vul* to a proprietary dataset which contains many additional variables (*i.e.*, EPSS, number of bugs, Kenna risk score). The proprietary dataset includes variables such as whether there are existing malware exploits and how extensively this vulnerability is discussed to help assess impact and risk. The proprietary data provide us with additional information and context for understanding the impact of a bug.

After splitting *Big-Vul* into the two datasets, we obtained $\approx 53K$ m triplets for the *example* generation and $\approx 5K$ n triplets for *question* generation (see Table I). We further filtered the *bigvul* dataset to include only sequences of 50-300 characters. The lower bound of 50 characters ensures that each sequence contains at least the function definition, while the upper bound of 300 characters balances the large number of experimental configurations against the computational cost of inference. Additionally, as shown in Figure 3, the majority of the data points fall within this range, and the CVSS score distribution remains the same proportion.

From this filtered dataset, we built the example and question prompts by randomly selecting m vulnerable code from the *example* dataset and selecting n vulnerable code from the *question* dataset. For example, for an experiment of $l =$

3 – *shots*, we selected code examples of $m = 900$ and $n = 300$ questions. We generate the set of interactions for each context type $p \in C_t$ by concatenating each example with its corresponding question. We selected a maximum of $l = 3$ due to the time-consuming nature of testing the model with additional examples and the input context size.

Our RQ_2 and RQ_3 require two testbeds, the first testbed uses stratified CVSS classification, and the second uses random CVSS extraction. For the stratified setting, we filtered data points with a sequence length between 50 and 300 characters. Then we classified both trained and test splits according to CVSS severity impact levels: none [0], low [0.1-3.9], medium [4-6.9], high [7-8.9], and critical [9-10](see Tab. II). The second testbed consists of a randomized sample drawn from the filtered dataset, including both questions and examples. This testbed is further partitioned by sequence length into two groups—50–100 (*short* sequence size) and 100–300 characters (*medium* sequence size)—to evaluate the impact of sequence size on performance.

TABLE I: Open-data *Big-Vul* size after curation

		Training	Test
Sequence Size	<i>medium</i>	43856	5524
	<i>short</i>	10425	1334
Documentation	Snippet + Summary	139607	17568
	Total	150908	18864

TABLE II: Counts per severity impact for train and test datasets with combined sequence size

Severity	Train Count	Test Count
None (0)	530	63
Low (0.1–3.9)	3853	472
Medium (4–6.9)	31386	3926
High (7–8.9)	14749	1889
Critical (9–10)	3763	508

Machine Configuration. We performed the experiments using 20.04 Ubuntu with an AMD EPYC 7532 32-core CPU, an A100 NVIDIA GPU with 40GB VRAM, and 1TB RAM. For the model inference process, we used HuggingFace and Pytorch [36], [49]. All models were loaded into the GPU to increase the inference time.

Open-source LLMs for Code. Our experiments utilized three types of LLMs. Firstly, *CodeLlama2*, is a specialized language model for programming tasks developed by Meta. The model can effectively identify potential security flaws and vulnerabilities in code by recognizing patterns indicative of weaknesses. The model also demonstrates proficiency in code interpretation, debugging, and documentation generation [46]. We used the Huggingface 7B and 13B model sizes for our experiments. Secondly, *Mistral*, is an open model that includes code generation capabilities reporting even better performance than Llama2 [2]. We used the 7B-size versions of these models via the HuggingFace `transformers` library and ran Mixtral through the Ollama runtime for our experiments [19], [32], [34], [49]. Finally, we evaluated OpenAI GPT4o-mini and GPT-OSS models [33], [35]. GPT4o-mini is our reference cloud model; GPT-OSS is an open-weight model that works with Ollama in a local environment. For all experiments, we set the following parameters: temperature of 0.001, 0.3, and 0.9, top-k of 50, and top-p of 0.92 for CodeLlama and Mistral models and default parameters for OpenAI models.

In-context configurations. Figure 2 illustrates the in-context setups used in our experiments. Following *Step ④* in Section III, we define three types of context. The first configuration, C_1 , consists of a set of examples determined by the number of shots, along with a direct query regarding a vulnerable code instance x' . The second configuration, C_2 , extends this setup by incorporating a system prompt that assigns the model the role of a vulnerability inspector, followed by the examples and the query. The third configuration, C_3 , further enhances the prompt by specifying the required output format in addition to the system prompt and examples. This design restricts the response to only the predicted CVSS score, preventing the inclusion of nonessential information.

B. Statistical Evaluation

We configured two experiments to predict the vulnerability severity score. The first is the stratified experiment, which consists of 600 context samples, each context sample with 3 shots. The second experiment is a random sample experiment that aims to evaluate the CVSS score prediction as a regular vulnerability inspector with a spontaneous snippet of code. Each experiment consists of 300 context samples.

Each context sample picks a random question to ask the severity score; the number of examples that relate to the question depends on the number of shots (Sec. V). For experiments using CodeLlama or Mistral with temperature 0.3 or 0.9, we collect the model response range by running each experiment 30 times. We aim to capture the model diversity outcome and also validate the default model’s parameters. Once the experiment is completed, we apply the following statistical methodology to evaluate our AI solution in terms of *in-context configuration effectiveness* and model accuracy. *Step 1:* Calculate the square difference error between predicted and expected score (gt_score) for each trial *Step 2:* Replace *unfeasible* values with the median of feasible predicted severity scores. An unfeasible value indicates that the generated text does not resemble an accepted severity value. Instead of predicting a concrete value, the model only generates a description of the code from the question prompt (*i.e.*, ‘Based on the provided code snippet alone, it is not possible or accurate to determine a CVSS score...’). *Step 3:* Compute the mean square error (*MSE*) for the 30 trials for each sample. To evaluate accuracy and in-context configuration effects, we compute the *mean MSE* among the 300 samples. In addition, we computed the following metrics. *Non-Feasible* constitutes the proportion of non-feasible or unanswered samples from the total of samples (*i.e.*, $\sum unfeasible/n$). *Out-Of-Range* embodies the number of trials when the model predicted a severity score outside the CVSS range (*i.e.*, predicted score > 10.0 or < 0.0). *Ex.Mean* and *Ex.Std.* are the mean and standard deviation of the score values in the input prompt examples. Finally, the *GT.Mean* and *GT.Std.* are the mean and standard deviation of the expected severity score (*i.e.*, ground truth scores).

RQ_2 and RQ_3 employed all the previous metrics. However, to answer RQ_2 we varied the context C to compare the performance, meanwhile, we focused on sequence length, and model variation to answer RQ_3 . The sequence length is calculated by the number of tokens used at the input prompt.

VII. RQ_1 EXPLORATORY DATA ANALYSIS

This section outlines the two datasets used in our case study, which serve as the basis for building prompts to evaluate the *open-source* models and configure test scenarios addressing our research questions. Since the proprietary dataset is confidential and has a limited number of data points, we supplemented it with an open-source dataset of vulnerabilities called *Big-Vul*.

We conducted two types of analysis on the *Proprietary* dataset and *Big-Vul*: a distribution comparison and an evaluation of the correlations between the datasets. These analyses aim to assess the naturalness of both datasets in generating our prompt testbed. By examining naturalness, we can determine whether the open-source dataset contains information comparable to the proprietary dataset, allowing us to use it interchangeably if needed.

A. Distribution Comparison

Industry Dataset Shape: The *Proprietary* data comprises three distinct datasets, each contributing valuable insights into vulnerabilities and risk assessment. The first dataset is the PSIRT CVE Vulnerability Dataset, which contains 5,520 data points. This data set includes detailed information such as an identifier (ID), associated CVE (Common Vulnerabilities and Exposures) and associated BugIDs. It also provides a title and description of each vulnerability, along with its severity rating, CVSS (Common Vulnerability Scoring System) score, and KCVSS (Kenna-adjusted CVSS) score. *Proprietary* dataset includes additional fields associated with the CWEs (Common Weakness Enumerations) and products affected by each vulnerability.

The second dataset is the Kenna Risk Dataset, comprising 5,372 data points. This dataset emphasizes risk factors and associated metadata for vulnerabilities. Key attributes include the CVE identifier and the Kenna Risk Score, which quantify risk. It also captures chatter indicators, reflecting the frequency of online discussions, and tags vulnerabilities for characteristics such as malware, remote code execution, and ease of exploitation. Additional fields flag vulnerabilities with active malware exploitation, breaches, and discussions that occur before the publication of the National Vulnerability Database (NVD).

The third dataset, the EPSS Dataset, is the largest, containing 221,346 data points. It focuses on predictive metrics and ranking of vulnerabilities. Key attributes include the CVE identifier, the EPSS (Exploit Prediction Scoring System) score—which estimates the likelihood of a vulnerability being exploited—and percentile rankings that position vulnerabilities within a broader exploitability landscape.

These datasets span 13 years, from 2010 to 2023, allowing a comprehensive longitudinal analysis of trends in vulnerability and risk. To provide additional context and validation, we compare the *Proprietary* data against the publicly available BigVul dataset, a well-known resource in vulnerability research [9].

Through this comparison, our objective is to uncover patterns, correlations, and insights that deepen our understanding of vulnerability, naturalness, risk prioritization, and precision. Together, these datasets form a solid foundation for our analysis, offering unique information dimensions critical to addressing our research objectives.

Fig. 4 shows the number of records that have each CVSS score for the *Proprietary* and *Big-Vul* datasets. Very few records have a CVSS score below 4. Compared to the *Proprietary* dataset, the *Big-Vul* dataset contains fewer records with high CVSS scores. However, the majority of CVSS scores in both datasets fall in the range of 4 to 8. The *Proprietary* dataset contains decimals, as shown in Fig. 4. Since *Big-Vul* has a larger number of data points than *Proprietary*, we group them by integer values. We notice that *Big-Vul* has a higher frequency on the CVSS score of 7 than *Proprietary*, however, the distribution overlaps and indicates a similar distribution.

TABLE III: Mean Squared Error calculated for each model with sequences *short* and *medium* size combined with 3-shots, varying the temperature T

Model	C	T	MSE Mean	Std. MSE	Gt. Mean	Gt. Std.	% Non Feasible
CodeLlama2 7B	C_1	0.001	7.91	13.01	6.14	1.82	0.00
		0.300	6.69	9.63	6.44	1.33	0.00
	C_2	0.001	8.36	12.74	6.13	1.83	0.50
		0.300	6.98	10.35	6.10	1.89	0.00
	C_3	0.001	10.00	16.21	6.06	1.88	4.00
		0.300	10.87	15.76	6.21	1.73	4.50
CodeLlama 13B	C_1	0.001	10.25	13.49	6.19	1.89	0.60
		0.300	12.31	16.49	6.11	1.99	5.00
	C_2	0.001	27.58	23.87	6.15	1.92	0.23
		0.300	26.42	24.60	6.08	1.94	0.68
	C_3	0.001	9.13	16.67	6.06	1.80	0.00
		0.300	10.81	16.47	6.16	1.98	0.50
gpt-oss	C_1	0.001	32.43	23.77	5.95	1.95	0.85
		0.300	34.65	23.62	6.14	1.89	2.80
	C_2	0.001	37.89	23.98	6.06	1.93	0.10
		0.300	35.39	22.82	5.88	1.88	0.90
	C_3	0.001	34.91	23.84	6.09	1.90	2.10
		0.300	37.28	24.86	6.06	2.07	2.00

*Best MSE in blue. Worst non-feasible proportions in red.

B. Distribution Correlation

Fig. 5 shows a heatmap of the relationship between the CVSS and KCVSS scores. The color of each box indicates the number of records with the respective CVSS and KCVSS scores. The heatmap on the right at Fig. 5 indicates a correlation between CVSS and KCVSS. KCVSS and CVSS are scores between 0 and 10.0, with the highest frequency at score 6. The heatmap on the right indicates a different correlation between *Kenna Risk* and CVSS.

Discussion. Comparing this *Proprietary* data to the publicly available *Big-Vul* dataset reveals overlapping CVSS score distributions, with notable alignment in the 4–8 range but variations such as *Big-Vul*’s higher frequency of CVSS 7 scores. This robust dataset suite provides nuanced insights into vulnerability patterns, risk prioritization, and exploitability. Although the CVSS-KCVSS relationship shows strong alignment, we observe a weaker correlation between CVSS and Kenna Risk scores.

VIII. RQ2 IN-CONTEXT CONFIGURATION EFFECTIVENESS

The purpose of evaluating the *in-context configuration* for predicting vulnerability severity is to observe how the model performs when we provide additional context explaining its role as a vulnerability evaluator using the context samples C_1 , C_2 , and C_3 . We started by testing the *open-source* model’s temperature impact, then varied the context and the number of shots.

Tab. III depicts the Avg. MSE of the severity score prediction and the standard deviation for each type of context C for *CodeLlama2*, *Mistral-7B*, and *gpt-oss* with sequences from 50 to 300 characters. *CodeLlama2-7B* outperforms other models, reaching the lowest MSE (6.69) and a 0.0% non-feasible rate. While *CodeLlama2-13B* performs similarly, its non-feasible rate degrades significantly at a higher temperature of 0.3 compared to its deterministic 0.001 setting. *gpt-oss* consistently underperforms across all conditions, with error rates roughly four to five times higher than the top-performing *CodeLlama2* configuration. We observed similar values for the

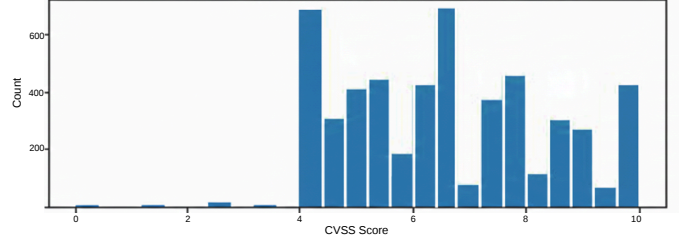
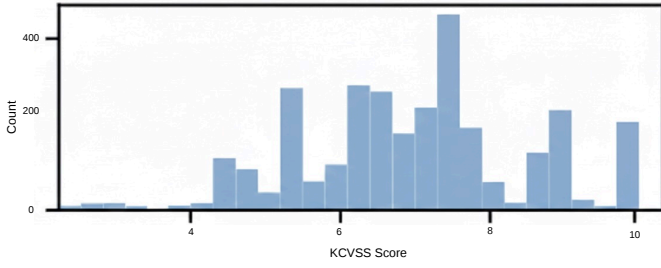


Fig. 4: Proprietary dataset frequencies: on the left, the KCVSS score, on the right, the assigned CVSS to the proprietary snippets.

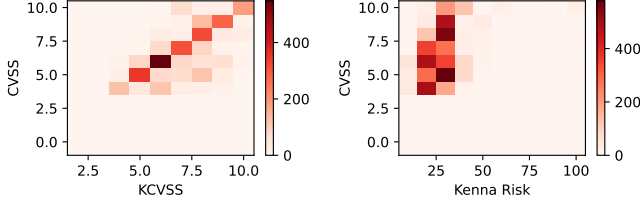


Fig. 5: Proprietary CVSS correlation with KCVSS and kenna risk, color represents the frequency.

Mixtral model with *gpt-oss* with the largest *MSE* values (see Appendix [40]).

The temperature calibration results indicate that *CodeLlama2-7B* achieves better predictive performance at $T = 0.3$ while maintaining a low non-feasible rate. Although *CodeLlama2-13B* performs slightly better with C_3 at $T = 0.001$, its performance at $T = 0.3$ remains comparable. Given that default temperature settings for *open-source* models are typically higher than $T = 0.001$, we selected a temperature of 0.3 for our empirical in-context effectiveness evaluation.

Table IV show a empirical evaluation for our *in-context* evaluation. For all three models, increasing the sequence size from *short* to *medium* and the number of iterations from $i = 0$ to $i = 3$ tends to reduce *MSE* and stabilize performance, with the best trade-offs between accuracy and feasibility appearing at larger sequences and higher iteration counts. Compared to *CodeLlama2-7B*, which shows the highest *MSE* and occasional *Non-Feasible* runs, *CodeLlama2-13B* narrows the gap to *GPT4o-mini* but still trails it, confirming *GPT4o-mini* as the strongest model overall in this setting. Table IV confirms the observed results of our temperature calibration experiment, and C_3 constitutes the best prompt configuration to predict the CVSS score among the *CodeLlama2* model and our reference model *GPT4o-mini*. Notably, we observe the highest *Non-Feasible* with the *CodeLlama2-13B* model for sequences with *medium* size and context C_1 . We also observe similar behavior with *CodeLlama2-7B*, with a slightly lower value of 2.34 for *medium* size sequences and C_1 . We avoided the *Mistral-7B* model as we observed *MSE* of ≈ 33 for all context configurations (see appendix [40]).

To address RQ₂, we also tested our models at $T = 0.9$ to capture the largest model outcome range. Table IV presents

TABLE IV: Mean Squared Error calculated for C_1, C_2 , and C_3 among shots $i = 0, 1, 3$ and temperature $T = 0.3$

Model	i	C	Seq. Size	MSE Mean	Std. MSE	% Non Feasible	Gt. Mean	Gt. Std.	Ex. Mean	Ex. Std.
<i>CodeLlama2-7B</i>	0	C_1	<i>short</i>	18.46	20.48	0.34	6.36	1.79	-	0.00
			<i>medium</i>	11.38	14.93	2.34	5.96	2.00	-	0.00
		C_2	<i>short</i>	30.16	24.20	0.00	6.36	1.71	-	0.00
			<i>medium</i>	14.22	18.17	0.00	6.16	1.71	-	0.00
		C_3	<i>short</i>	10.18	12.80	0.00	6.47	1.92	-	0.00
			<i>medium</i>	11.55	12.50	0.00	5.93	1.82	-	0.00
	1	C_1	<i>short</i>	8.24	11.68	0.00	6.42	1.72	6.16	0.00
			<i>medium</i>	7.00	10.05	0.34	6.07	1.87	6.00	0.00
		C_2	<i>short</i>	8.45	11.97	0.00	6.43	1.90	6.45	0.00
			<i>medium</i>	7.39	9.45	0.00	6.17	1.92	5.91	0.00
		C_3	<i>short</i>	6.99	9.36	0.00	6.32	1.75	6.11	0.00
			<i>medium</i>	6.96	9.72	0.00	6.28	1.77	6.30	0.00
<i>CodeLlama2-13B</i>	0	C_1	<i>short</i>	8.76	12.92	0.67	6.47	1.80	6.16	1.32
			<i>medium</i>	8.94	11.56	1.36	6.09	1.69	6.18	0.00
		C_2	<i>short</i>	9.85	13.96	0.00	6.31	1.78	6.16	1.28
			<i>medium</i>	15.49	21.26	0.00	6.01	1.87	6.14	0.00
		C_3	<i>short</i>	5.86	7.64	0.00	6.25	1.70	6.19	1.42
			<i>medium</i>	12.30	12.83	0.00	5.90	1.76	-	0.00
	1	C_1	<i>short</i>	31.63	22.70	1.00	6.26	1.81	-	0.00
			<i>medium</i>	21.39	20.00	3.36	5.97	1.77	-	0.00
		C_2	<i>short</i>	26.90	24.02	0.00	6.25	1.84	-	0.00
			<i>medium</i>	25.25	23.67	0.00	6.10	1.76	-	0.00
		C_3	<i>short</i>	9.91	12.09	0.00	6.05	1.79	-	0.00
			<i>medium</i>	10.57	12.75	0.00	5.92	1.86	-	0.00
	3	C_1	<i>short</i>	13.74	18.33	0.00	6.24	1.87	6.22	0.00
			<i>medium</i>	9.04	12.82	0.00	6.07	1.80	5.80	0.00
		C_2	<i>short</i>	19.95	21.38	0.00	6.26	1.89	6.16	0.00
			<i>medium</i>	13.29	18.85	0.00	5.98	1.91	5.93	0.00
		C_3	<i>short</i>	5.80	8.04	0.00	6.41	1.73	6.41	0.00
			<i>medium</i>	6.07	9.45	0.00	6.20	1.80	5.95	0.00
<i>GPT4o-mini</i>	0	C_1	<i>short</i>	13.89	18.37	0.00	6.26	1.74	6.14	1.36
			<i>medium</i>	11.04	15.71	0.00	5.84	1.86	5.88	1.38
		C_2	<i>short</i>	17.54	21.54	0.00	6.26	1.75	6.21	1.37
			<i>medium</i>	11.31	15.76	0.00	6.06	1.79	5.89	1.41
		C_3	<i>short</i>	4.90	6.58	0.00	6.09	1.81	6.16	1.36
			<i>medium</i>	6.49	8.28	0.00	5.99	1.97	5.98	1.37
	1	C_1	<i>short</i>	18.65	22.82	0.00	6.21	1.93	-	0.00
			<i>medium</i>	8.62	11.26	0.00	6.02	1.92	-	0.00
		C_2	<i>short</i>	18.22	20.14	0.00	5.85	1.86	-	0.00
			<i>medium</i>	8.95	10.40	0.00	6.11	1.70	-	0.00
		C_3	<i>short</i>	3.44	4.77	0.00	6.26	1.87	-	0.00
			<i>medium</i>	3.79	5.67	0.00	6.12	1.85	-	0.00
	3	C_1	<i>short</i>	13.79	19.94	0.00	6.29	1.83	0.00	0.00
			<i>medium</i>	9.82	14.10	0.00	6.04	1.93	5.93	0.00
		C_2	<i>short</i>	11.10	17.92	0.00	6.15	1.86	6.12	0.00
			<i>medium</i>	7.85	10.99	0.00	5.98	1.93	5.92	0.00
		C_3	<i>short</i>	3.46	5.38	0.00	6.34	1.77	6.15	0.00
			<i>medium</i>	3.67	5.38	0.00	6.01	1.78	5.77	0.00
<i>GPT4o-mini</i>	0	C_1	<i>short</i>	7.19	9.72	0.00	6.42	1.72	5.92	0.00
			<i>medium</i>	5.85	8.77	0.00	6.02	1.76	6.09	1.34
		C_2	<i>short</i>	7.14	10.71	0.00	6.15	1.86	6.11	0.00
			<i>medium</i>	5.41	7.06	0.00	6.09	1.67	5.91	1.35
		C_3	<i>short</i>	3.08	4.58	0.00	6.25	1.79	5.89	0.00
			<i>medium</i>	3.17	4.78	0.00	6.07	1.82	5.98	1.40

*Best *MSE* in blue-bold. Worst non-feasible proportions in red.

the *MSE* for *CodeLlama2* and *Mistral-7B* models. We observe a higher *Non-Feasible* proportion for C_1 than C_2 for both models *CodeLlama2-7B* and *Mistral-7B* with zero shots; for example, *CodeLlama2-7B* has a *Non-Feasible* value of 16% and 80.33% for *Mistral-7B* with C_1 and 0% with C_2 .

In comparison, we observe no big difference in the accuracy *MSE* between using C_1 and C_2 . *CodeLlama2-7B* with 3-shots and C_1 has an *MSE* of 14.92 and a Std. of 10.65 while the same configuration as C_2 has a *MSE* of 14.05 and a Std. of

TABLE V: Mean Squared Error calculated for each model and sequence sizes with temperature of $T = 0.9$

Model	C	Seq. Size	l	Avg. MSE	Std. MSE	Non Feasible	Out-Of-Range	Ex. Mean	Ex. Std.	Gt. Mean	Gt. Std.
CodeLlama2-7B	C ₁	medium	0	19.88	6.51	22.61%	0.23%	-	-	6.08	1.79
			1	7.65	6.52	0.00%	0.10%	6.08	1.96	6.09	1.85
			2	8.20	6.68	0.00%	0.01%	5.95	1.89	6.19	1.84
		short	3	9.01	7.20	0.00%	0.04%	6.01	1.91	6.12	1.86
			0	32.08	4.99	16.00%	0.07%	-	-	6.11	1.70
			1	11.53	9.27	0.00%	0.19%	6.13	2.08	6.13	2.08
	C ₂	medium	2	14.19	10.06	0.00%	0.02%	6.25	1.93	6.34	1.90
			3	14.92	10.65	0.00%	0.01%	6.12	1.88	6.39	1.89
			0	31.17	11.06	0.00%	0.15%	-	-	6.15	1.70
		short	1	15.98	12.22	0.00%	0.06%	6.29	1.89	6.12	1.93
			2	15.23	11.93	0.00%	0.01%	6.22	1.84	6.33	1.78
			3	14.05	11.39	0.00%	0.01%	6.07	1.93	6.17	1.77
Mistral-7B	C ₁	medium	0	28.73	1.01	41.69%	1.10%	-	-	6.22	1.85
			1	28.85	5.22	11.11%	1.21%	6.14	2.02	6.09	1.87
			2	26.26	5.26	10.10%	2.40%	6.02	1.90	5.95	1.82
		short	3	27.23	5.03	11.71%	2.58%	5.94	1.97	6.10	1.82
			0	25.97	1.98	80.33%	0.92%	-	-	6.25	1.80
			1	31.98	4.49	15.72%	0.60%	6.23	1.85	6.11	1.87
	C ₂	medium	2	35.31	2.50	27.27%	0.10%	6.22	1.87	6.33	1.68
			3	31.48	3.17	35.25%	1.62%	6.11	1.88	6.38	1.75
			0	37.50	0.71	44.30%	0.44%	-	-	6.20	1.81
		short	1	21.72	3.67	47.80%	0.90%	6.32	1.77	6.20	1.87
			2	27.60	1.80	60.61%	1.13%	6.26	2.02	6.08	1.78
			3	30.89	1.14	59.93%	1.55%	6.20	1.85	6.22	1.97
CodeLlama2-13B	C ₁	medium	0	10.55	9.98	0.00%	5.72%	-	-	6.17	1.68
			1	10.15	9.32	0.00%	0.73%	6.06	1.77	6.12	1.84
			2	9.27	8.66	0.00%	0.01%	6.07	1.87	5.96	1.89
		short	3	10.50	9.22	0.00%	0.00%	5.92	1.94	6.20	1.86
			0	16.19	13.71	0.00%	8.44%	-	-	6.36	1.82
			1	15.02	12.43	0.00%	0.00%	6.21	1.98	6.26	1.84
	C ₂	medium	2	14.36	11.72	0.00%	0.00%	6.24	1.81	6.03	1.78
			3	15.15	12.83	0.00%	0.00%	6.31	1.85	6.22	1.83
		short	0	10.55	9.98	0.00%	5.72%	-	-	6.17	1.68
			1	10.15	9.32	0.00%	0.73%	6.06	1.77	6.12	1.84
			2	9.27	8.66	0.00%	0.01%	6.07	1.87	5.96	1.89
			3	10.50	9.22	0.00%	0.00%	5.92	1.94	6.20	1.86

*Best MSE in blue-bold. Worst non-feasible and out-of-range proportions in red.

11.39.

Mistral-7B obtains the best MSE of 21.72 with C_2 ; however, the proportion of non-feasible (47.80%) is high. We notice that the role and task description of C_2 improves the non-feasible proportion to the zero-shot configuration by observing 80.33% with C_1 and 44.3% with C_1 using *Mistral-7B*. Similarly, *CodeLlama2-7B* has a Non-Feasible proportion of 16% zero shot and C_1 , and 0% with zero-shot and C_2 .

Discussion. Our findings reveals positive impact of using C_3 compared to C_1 and slight improvement of using C_3 compared with C_2 when using *CodeLlama2* model with 0.3 temperature. Comparing the same context samples, we observe an impact on the number of shots; *Accuracy jumps* observed from zero-shot to 1-shot indicate that LLMs obtain more effective guidance by providing examples rather than explaining the role (*i.e.*, the CVSS is a severity value that ranges between 0 to 10) and describing the task (*i.e.*, your roles is to examine the code and predict a score). We observe no significant difference between C_1 and C_2 at 0.9 temperature. Instead, C_2 positively impacts the prediction by reducing the Non-Feasible proportion in the predicted score. However, the model potentially produces inaccurate predictions outside the correct score range.

IX. RQ₃ OPEN-SOURCE LLMs ACCURACY

The MSE alone does not provide sufficient reliability and therefore requires a complementary measure. To address this, we use Non-Feasible, which captures whether at least one valid answer emerges within 30 attempts. For example, *Mistral-7B* with zero-shot prompting and *short* sequence length achieves a MSE of 25.97, lower than that obtained with *medium* sequences under the same prompting condition. However, this improvement coincides with a substantially higher rate of feasible answers: Non-Feasible increases from 41.69% for *medium* sequences to 80.33% for *short* sequences. We run experiments on each model *CodeLlama2-7B* and *CodeLlama2-*

13B, *Mistral-7B* with C_1 varying the prompt size and the number of shots. Tab. IV and Tab. V illustrate the regression and standard deviation metrics computed for the experiment. We observe that *CodeLlama2* models outperform *Mistral-7B* in all sequence sizes and have lower Non-Feasible rates.

In contrast, the *CodeLlama2* models have a lower number of Non-Feasible responses. We observe that *CodeLlama2* on zero-shot does not have enough context in the score range. Therefore, the Out-Of-Range percentage increases to 8.44% for a sequence length of 100. Note that the best setup is the *CodeLlama2-7B* model size with just one-shot and sequence size of 300, obtaining an Avg. MSE of 7.65 and a Std. of 6.52 with a 0.10% Out-Of-Range values. Similar behavior is observed with *CodeLlama2-7B* with a sequence size of 300 and two shots observing an Avg. MSE of 9.27 and a Std. of 8.66.

All models, except *CodeLlama2-13B*, produce a significant number of Non-Feasible responses with a temperature of 0.9. Among them, *Mistral-7B* in the zero-shot setting performs the worst, with 80.33% of its responses being Non-Feasible. This result indicates that the model requires more context and a clearer description of the expected output to generate acceptable outcomes. In contrast, *CodeLlama2-13B* shows a consistent ability to predict feasible CVSS values without the need for additional context. A similar pattern is observed with *CodeLlama2-7B*; however, its performance depends on the specific example provided. For example, when evaluated with C_1 in the zero-shot setting, *CodeLlama2-7B* occasionally fails to generate a valid value, highlighting its sensitivity to the input example.

Figure 6 presents confusion matrices summarizing the severity classification performance of the three evaluated models (*CodeLlama2-7B*, *CodeLlama2-13B*, and *GPT4o-mini*) on the held-out test set, with rows denoting true severity and columns the predicted severity.

The *CodeLlama2-7B* model (left) concentrates predictions in the “Medium” column, leading to high correct rates for true “Medium” cases but systematic under-prediction of “None” and “Low” severities. This tendency to default to “Medium” results in fewer extreme false positives, but at the cost of reduced discrimination among lower-severity classes. For *CodeLlama2-13B* (center), has a tendency to predict scores to the “None” category. The matrix also reveals a notable misclassification in the most extreme “Critical” category, suggesting conservative behavior when assigning the highest severity.

The *GPT4o-mini* model (right) shows comparatively stronger concentrations for the “None” and “Medium” classes, indicating a slightly better alignment between “Medium” and predicted labels in these regions. At the same time, the model fails to predict “Critical” labels, producing “None” or “Medium” classifications. Similarly to *CodeLlama2-13B*, the model almost never predicts “Critical,” suggesting a cautious stance toward the highest severity assignments.

Discussion. We observe that *CodeLlama2* models outperform in predicting a score value given a prompt with vulner-

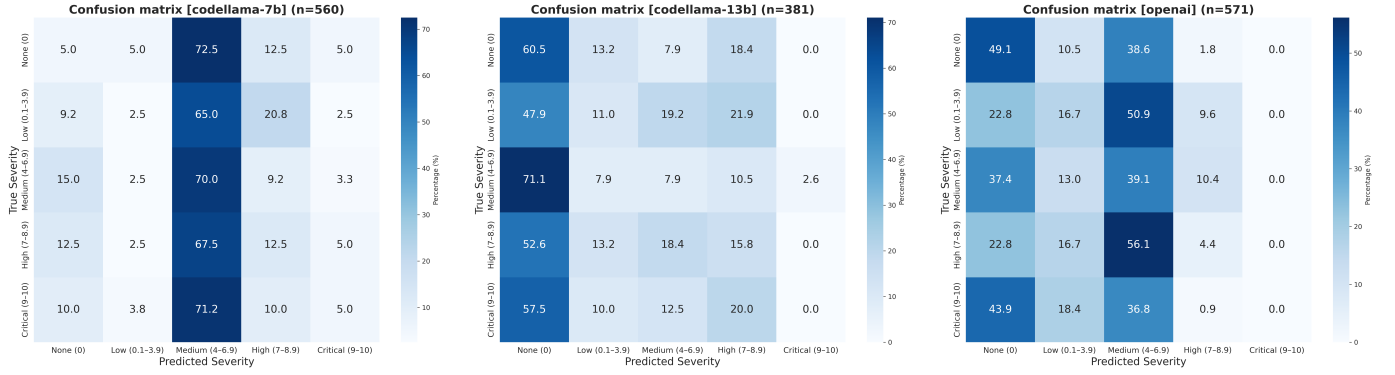


Fig. 6: Confusion matrix for stratified CVSS classification using *CodeLlama2-7B*, *CodeLlama2-13B* and *GPT4o-mini* with $n = 600$ datapoints.

able code and examples; *CodeLlama2* models tend to predict a value and then explain why the value is valid. *CodeLlama2* models increase the verbosity of the generated text as the size of the model increases. Consequently, the *Std.* for 13B is larger than the ones for 7B. For example, *CodeLlama2-7B*, with a three-shot sequence size of 300, has a variation of 10.65, while the same experiment for 13B has a variation of 12.83. In contrast, *Mistral-7B* is prone to summarize code execution before predicting the severity score.

We observe a small difference between the model sizes’ performance. In fact, the small *CodeLlama2-7B* has better accuracy in completing the scoring prediction tasks with a few *Out-Of-Range* values. We highlight that the *MSE* values are comparable between LLMs and experiments, since we sampled from the same *Big-Vul* data set and observe very close values between experiments for *Ex.Mean*, *Ex.Std.*, *GT.Mean*, and *GT.Std.* The matrix structure suggests that *CodeLlama2-7B* is less calibrated across the full severity range, effectively blending several neighboring classes into a single dominant prediction.

X. RELATED WORK

Prior work on learning-based vulnerability analysis has focused on traditional machine learning or pre-LLM deep models for detection and severity estimation, for example, linear models and random forests for CVSS prediction from textual descriptions, or CNN/RNN/Transformer-based models trained on code-level datasets such as *Big-Vul* [9]. More recently, deep vulnerability-focused models such as VulDeePecker [26], VulBERTa [16], and LineVul [13] have shown that specialized architectures and pretraining objectives can improve code-level vulnerability detection, but they still require task-specific training and do not directly address CVSS regression. In contrast, a growing body of work now studies how general-purpose LLMs can be adapted to security tasks, including LLM-assisted static analysis (IRIS) [24], LLM-based data-flow reasoning (LLMDFA) [48], CVE/CVSS classification [31], LLM-based vulnerability detection, classification, and repair [14], [23], and critiques of current ML benchmarking practices for vulnerability detection [39]. These studies highlight both the promise and the limitations of LLMs for finding bugs and reasoning

about program behavior, yet they primarily treat vulnerability detection as a classification problem and often focus on cloud-hosted or fine-tuned models. Our work complements this line by targeting severity regression from vulnerable code using in-context learning with locally deployable, open-source code LLMs, and by quantifying their calibration and robustness under different prompt configurations.

XI. THREATS TO VALIDITY

Our study operates on *Big-Vul*’s function-level representation of vulnerabilities, which abstracts away much of the surrounding project and operational context. As a result, our results should be interpreted as performance in these isolated snippets rather than in fully context-rich real-world cases.

Construct Validity Our study assumes that function-level snippets from *Big-Vul* contain sufficient information to predict CVSS severity. However, recent work shows that method-level vulnerability labels in *Big-Vul* can be noisy and may not capture the full vulnerability context, which can bias models towards learning superficial correlations rather than true vulnerability semantics [37], [38]. We partially mitigate this by filtering for longer snippets with descriptions, but our results should still be interpreted as performance on *Big-Vul*’s labeling rather than on arbitrary real-world code. In addition, we predict only the aggregate CVSS score, not its individual submetrics, which limits interpretability and actionability for practitioners.

Internal Validity Our results depend on specific choices for prompt format, number of shots, sequence size, and decoding temperature. We explore several configurations and, in the revised version, add a temperature calibration experiment comparing $T = 0.001$ and $T = 0.3$ with context C_3 , adopting $T = 0.3$ as a low-temperature setting that balances error and output diversity. However, better configurations may exist, and our reported numbers should be seen as indicative rather than optimal. We also clarify that high-temperature, multi-sample runs are used only for analyzing variability, while realistic usage should rely on a single, low-temperature prediction per query.

External Validity We evaluate only C/C++ code from *Big-Vul* and open-source models in the 7B–13B range (*CodeL-*

lama2, Mistral). Our findings may not be generalized to other languages, vulnerability types, larger or more recent models (e.g., Llama 3, Qwen 2.5), or different industrial environments. Proprietary datasets are used solely for exploratory distribution comparison, so we cannot claim end-to-end effectiveness on those systems. Moreover, recent work warns that standard vulnerability benchmarks can overestimate real-world performance due to benchmark design and spurious correlations [37], [38].

Data Contamination and Benchmarking Bias Big-Vul is built from public CVEs and CVSS scores, which likely appear in the pretraining data of our open-source models. This creates a risk of data contamination, where some examples may be partially memorized, increasing performance estimates [4], [5]. Because we cannot audit the pretraining corpora, we treat our numbers as an upper bound under possible contamination rather than a clean measure of out-of-distribution generalization, in line with recent discussions on contamination in LLM benchmarks [5].

XII. CONCLUSIONS AND FUTURE WORK

We applied our approach as our first experience in an industry experience in a real environment. The motivation relies on creating a private and secure environment for using proprietary data and software code. Using *open-source* small and locally deployed models. This *open-source* configures the main alternative to private options such as GitHub Copilot or OpenAI models. Our findings show a good impact on the use of *open-source* models in achieving this.

The use of an in-context learning approach for local models further enhances this capability by enabling them to adapt dynamically to each company’s specific security policies and requirements. Our approach demonstrates that acceptable performance can be achieved even with small models such as *CodeLlama2-7B*, striking a balance between computational efficiency and effective policy enforcement.

Limitations. We predict only the aggregate CVSS v3.1 base score from C/C++ snippets, not individual submetrics or richer contextual factors (e.g., infrastructure, asset criticality, social signals). Our experiments are constrained to *Big-Vul* and to 7B–13B open-source models (*CodeLlama2-7B/CodeLlama2-13B*, *Mistral-7B*) that fit single-GPU, *on-premise* deployments, so larger recent open or proprietary models (e.g., Llama 3, Qwen 2.5 32B, DeepSeek R1) remain outside our scope, and we do not offer a comprehensive LLM benchmark.

L₁: Datasets include obsolete or poor quality vulnerability description. We observe a similar distribution and behavior between the *Proprietary* and open datasets. However, the data can sometimes be outdated, including information from previous years. Since vulnerabilities and attacks on software systems evolve more rapidly than their corresponding reports, the available data points for training and testing models are often insufficient to meet the demands of modern cybersecurity challenges.

L₂: LLMs do not improve accuracy after 3–shots. We observe a noticeable improvement in *MSE* predictions and

feasibility at the one-shot setting. However, beyond two shots, LLMs do not appear to gain sufficient additional context to meaningfully improve accuracy. Instead, higher accuracy is achieved with *C₃* when the desired outcome format is explicitly specified. We did not evaluate additional shots due to the computational cost and the marginal improvement observed between the two-shot and three-shot settings.

L₃: LLMs can assist security analysts in identifying vulnerable code before deployment. Small and medium-sized companies can benefit from using *open-source* models while keeping their data custody and saving costs due to local execution fitting on medium GPU requirements. However, predicting severity requires larger context variables (e.g., third-party libraries, vulnerability social media information, similar vulnerabilities information) and in-context configurations to the environmental customer variable (e.g., infrastructure, data protection policies, data profile). Both *CodeLlama2* and *Mistral-7B* LLMs tend to generate explanations about the code function. The explanation can be used to calibrate the model and also to track the vulnerability report.

Future work. An immediate extension is to move beyond a single-target regression setup and jointly model CVSS alongside related risk signals such as Kenna Risk and EPSS, enabling multi-target inference that better reflects real-world prioritization pipelines. Decomposing the task to predict CVSS submetrics (e.g., attack vector, privileges required, user interaction, impact metrics) in addition to the aggregate score could also yield more interpretable and actionable guidance for analysts. On the modeling side, evaluate stronger recent open-weight models and selected proprietary systems under realistic deployment constraints—for example, carefully comparing 7B–13B on-premise models against larger (e.g., 32B–70B).

REFERENCES

- [1] Anonymous. Gemini: Openai’s multimodal language model. *arXiv preprint arXiv:2306.12345*, 2023.
- [2] Anonymous. Mistral: A comprehensive and efficient ai model for nlp. *arXiv preprint arXiv:2306.03034*, 2023.
- [3] Anthropic. Claude: An ai assistant for natural language understanding. *arXiv preprint arXiv:2310.04620*, 2023.
- [4] A. Author. An overview of data contamination: The causes, risks, signs, and implications. *HolisticAI Blog*, 2024.
- [5] B. Author and C. Author. Does data contamination detection work (well) for llms? a survey. In *Findings of NAACL*, 2025.
- [6] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–114, Washington DC USA, July 2010. ACM.
- [7] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.
- [8] CyberScoop. Infosec pros: We need cvss, warts and all. <https://cyberscoop.com/cvss-criticism-cve-nvd-nist-epss/>, 2025. Accessed Feb. 2026.
- [9] J. Fan, Y. Li, S. Wang, and T. N. Nguyen. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 508–512, Seoul Republic of Korea, June 2020. ACM.
- [10] Y. Fang, Y. Liu, C. Huang, and L. Liu. FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *PLOS ONE*, 15(2):e0228439, Feb. 2020.

- [11] FIRST.org. Common vulnerability scoring system version 3.1: Specification document. <https://www.first.org/cvss/specification-document>, 2021. Accessed: 06-26-2024.
- [12] FIRST.org. Cvss v3.1 user guide. Technical report, Forum of Incident Response and Security Teams, 2021.
- [13] M. Fu and C. Tantithamthavorn. LineVul: a transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 608–620, Pittsburgh Pennsylvania, May 2022. ACM.
- [14] M. Fu, C. Tantithamthavorn, V. Nguyen, and T. Le. ChatGPT for Vulnerability Detection, Classification, and Repair: How Far Are We?, Oct. 2023. arXiv:2310.09810 [cs].
- [15] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, and X. Liao. Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, Lisbon Portugal, Feb. 2024. ACM.
- [16] H. Hanif and S. Maffei. VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2022. arXiv:2205.12424 [cs].
- [17] M. S. Hoque, N. Jamil, N. Amin, and K.-Y. Lam. An Improved Vulnerability Exploitation Prediction Model with Novel Cost Function and Custom Trained Word Vector Embedding. *Sensors*, 21(12):4220, June 2021.
- [18] Y. Huang, Y. Liu, R. Thirukovalluru, A. Cohan, and B. Dhingra. Calibrating long-form generations from large language models, 2024.
- [19] Hugging Face. Mixtral – hugging face transformers documentation. https://huggingface.co/docs/transformers/en/model_doc/mixtral, 2023. Accessed Feb. 2026.
- [20] IBM Security. Cost of a data breach report 2024. <https://newsroom.ibm.com/2024-07-30-ibm-report-escalating-data-breach-disruption-pushes-costs-to-new-highs>, 2024. Accessed Feb. 2026.
- [21] IBM Security and Ponemon Institute. Cost of a data breach report 2024. <https://wp.table.media/wp-content/uploads/2024/07/30132828/Cost-of-a-Data-Breach-Report-2024.pdf>, 2024. Accessed Feb. 2026.
- [22] J. Jacobs, S. Romanosky, B. Edwards, I. Adjerid, and M. Roytman. Exploit Prediction Scoring System (EPSS). *Digital Threats: Research and Practice*, 2(3):1–17, Sept. 2021.
- [23] A. Khare, P. Gupta, B. Sharma, and B. Ray. Understanding the effectiveness of large language models in detecting security vulnerabilities. *Empirical Software Engineering*, 2025. Preprint.
- [24] X. Li, Y. Zhang, J. Chen, S. Wang, and T. N. Nguyen. Iris: Llm-assisted static analysis for detecting security vulnerabilities. In *Proceedings of the 47th International Conference on Software Engineering*, 2025. To appear.
- [25] Y. Li, S. Wang, and T. N. Nguyen. Vulnerability Detection with Fine-grained Interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 292–303, Aug. 2021. arXiv:2106.10478 [cs].
- [26] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In *Proceedings 2018 Network and Distributed System Security Symposium*, 2018. arXiv:1801.01681 [cs].
- [27] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. Raffel. Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning, Aug. 2022. arXiv:2205.05638 [cs].
- [28] Z. Liu, Q. Liao, W. Gu, and C. Gao. Software Vulnerability Detection with GPT and In-Context Learning. In *2023 8th International Conference on Data Science in Cyberspace (DSC)*, pages 229–236, Hefei, China, Aug. 2023. IEEE.
- [29] M. Luo, X. Xu, Y. Liu, P. Pasupat, and M. Kazemi. In-context learning with retrieved demonstrations for language models: A survey, 2024.
- [30] D. E. Mann and S. M. Christey. Towards a common enumeration of vulnerabilities. White paper, The MITRE Corporation, 1999.
- [31] F. Marchiori, D. Donadel, and M. Conti. Can llms classify cves? investigating llms capabilities in computing cvss vectors, 2025.
- [32] Ollama. mixtral:8x7b model card. <https://ollama.com/library/mixtral:8x7b>, 2024. Accessed Feb. 2026.
- [33] Ollama. gpt-oss model card. <https://ollama.com/library/gpt-oss>, 2025. Accessed Feb. 2026.
- [34] Ollama. Ollama documentation. <https://docs.ollama.com>, 2026. Accessed Feb. 2026.
- [35] OpenAI. Gpt-4o mini model. <https://developers.openai.com/api/docs/models/gpt-4o-mini>, 2024. Accessed Feb. 2026.
- [36] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [37] N. Risse, J. Liu, and M. Böhme. On benchmarking in machine learning for vulnerability detection. *arXiv*, 2024.
- [38] N. Risse, J. Liu, and M. Böhme. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection. In *Proceedings of the ACM on Software Engineering*, 2025.
- [39] N. Risse, J. Liu, and M. Böhme. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection, 2025.
- [40] D. Rodriguez-Cardenas, D. N. Palacio, A. Schmedding, Y. Lu, B. Hudson, C. Gourley, M. Roytman, C. Shenefiel, E. Smiri, and D. Poshvanyk. Artifacts for “on predicting vulnerability severity using in-context learning: An industrial case study”. <https://anonymous.4open.science/status/in-context-vul-severity-34CC>, 2025. Experiment code and datasets.
- [41] R. L. Russell, L. Kim, L. H. Hamilton, T. Lazovich, J. A. Harer, O. Ozdemir, P. M. Ellingwood, and M. W. McConley. Automated Vulnerability Detection in Source Code Using Deep Representation Learning, Nov. 2018. arXiv:1807.04320 [cs, stat].
- [42] C. Sabottke, O. Suciu, and T. Dumitras. 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, SEC’15*, page 1041–1056, USA, 2015. USENIX Association.
- [43] J. Spring, E. Hatleback, A. D. Householder, A. Manion, and D. Shick. Prioritizing vulnerability response: A stakeholder-specific vulnerability categorization. Carnegie Mellon University, Software Engineering Institute, White Paper, Dec 2019. Accessed: 2024-Jul-9.
- [44] M. Tan, M. A. Merrill, V. Gupta, T. Althoff, and T. Hartvigsen. Are language models actually useful for time series forecasting?, 2024.
- [45] I. Tenable. Vulnerability priority rating, 2023. Accessed: 2024-07-15.
- [46] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [47] VMware. Understanding the kenna security vulnerability risk score. Whitepaper, VMware, Inc., 2021. A part of Cisco.
- [48] Y. Wang, H. Liu, J. Gu, and B. Xu. Llmdfa: Analyzing dataflow in code with large language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024.
- [49] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.
- [50] B. Wu and F. Zou. Code Vulnerability Detection Based on Deep Sequence and Graph Models: A Survey. *Security and Communication Networks*, 2022:1–11, Oct. 2022.
- [51] C. Xiao, A. Sarabi, Y. Liu, B. Li, M. Liu, and T. Dumitras. From patching delays to infection symptoms: using risk profiles for an early discovery of vulnerabilities exploited in the wild. In *Proceedings of the 27th USENIX Conference on Security Symposium, SEC’18*, page 903–918, USA, 2018. USENIX Association.
- [52] Y. Zhang, J. Li, et al. Survey of source code vulnerability analysis based on deep learning. *Computers & Security*, 2024. Preprint; emphasizes binary classification of source code into “vulnerable” or “non-vulnerable”.