Out of Distribution, Out of Luck: How Well Can LLMs Trained on Vulnerability Datasets Detect Top 25 CWE Weaknesses?

Yikun Li*, Ngoc Tan Bui*, Ting Zhang*, Martin Weyssow*, Chengran Yang*, Xin Zhou*, Jinfeng Jiang*, Junkai Chen*, Huihui Huang*, Huu Hung Nguyen*, Chiok Yew Ho§, Jie Tan‡, Ruiyin Li¶, Yide Yin†, Han Wei Ang†, Frank Liauw†, Eng Lieh Ouh*, Lwin Khin Shar*, David Lo*

*Singapore Management University
Singapore, Singapore
§Chinese University of Hong Kong
Hong Kong, China
‡University of Groningen
Groningen, The Netherlands
¶Wuhan University
Wuhan, China
†GovTech
Singapore, Singapore

Abstract

Automated vulnerability detection research has made substantial progress, yet its real-world impact remains limited. Current vulnerability datasets suffer from issues including label inaccuracy rates of 20-71%, extensive duplication, and poor coverage of critical Common Weakness Enumeration (CWE) types. These issues create a significant "generalization gap" where models achieve misleading self-testing accuracies (i.e., accuracy measured on held-out data from the same dataset used for training) by exploiting spurious correlations rather than learning true vulnerability patterns. Our analysis reveals that many models experience substantial performance drops of up to 40.6% when evaluated on independent data, underperforming random guessing.

To address these limitations, we present a comprehensive three-part solution. First, we introduce a manually curated *test dataset*, BenchVul, covering the MITRE Top 25 Most Dangerous CWEs [17]. Second, we construct a high-quality *training dataset*, TitanVul, comprising 35,045 functions by aggregating seven public sources and applying rigorous deduplication and validation using a novel multi-agent LLM framework. Third, we propose a Realistic Vulnerability Generation (RVG) framework, which synthesizes context-aware vulnerability examples for underrepresented but critical CWE types through simulated development workflows.

Our evaluation highlights the strengths of each component in closing the generalization gap. First, BenchVul shows the limitations of self-testing: models trained on existing datasets, such as BigVul and PrimeVul, experience performance drops on BenchVul (e.g., from 0.776 to 0.519 and from 0.567 to 0.337, respectively). Second, training models on TitanVul demonstrates improved generalization, with model performance increasing from 0.584 when evaluated on the same dataset (self-testing) to 0.767 when tested on BenchVul. Third, supplementing TitanVul with RVG-generated data yields further gains, increasing model performance by 14.0% to 0.874. Code and data are available at: https://github.com/yikun-li/TitanVul-BenchVul.

1 Introduction

Automated vulnerability detection is a popular area of software engineering research [2, 5, 9]. A recent survey reported that 88% of studies in machine learning for vulnerability detection (ML4VD) approach the problem as function-level classification: given a function's source code, the task is to determine whether it contains a vulnerability [4, 16, 23]. However, prior work has identified significant data quality issues in widely used vulnerability datasets, including high rates of label inaccuracy (20-71%) and extensive data duplication [3–5, 5, 16, 23]. In addition, as shown in Section 2, available datasets are highly fragmented, imbalanced, and often contain outdated or incorrect CWE labels. Moreover, many so-called *vulnerable functions* are not self-contained; their vulnerability can only be understood by analyzing the external context, which is absent in function-level analysis [23].

Such dataset issues artificially inflate model performance when evaluations rely on self-testing, i.e., assessments using subsets of the same datasets used in training. Inflated accuracy arises from models capturing dataset-specific biases rather than learning genuine vulnerability patterns [23], causing a significant gap between reported accuracy and real-world effectiveness. This generalization gap undermines reliable model comparisons and assessments of dataset quality. We notice three challenges:

Challenge I: Unreliable Evaluation Due to Overfitting Currently, researchers rely primarily on self-testing evaluations, which use test samples drawn from the same datasets used for training. As these datasets often contain duplicated samples and labeling issues, models may achieve high self-testing accuracies by memorizing dataset-specific artifacts rather than learning generalizable vulnerability patterns. Consequently, the actual capability of models to detect vulnerabilities in realistic scenarios remains uncertain [23].

Challenge II: Poor-Quality Training Data at Scale Widely-used vulnerability datasets suffer from low data quality, including high rates of noise, irrelevant code changes, refactoring, and non-security-related fixes [4, 23]. Although these datasets are large, they

often lack vulnerabilities and their corresponding fixes that are self-contained at the function level. This limitation prevents models from learning robust and generalizable vulnerability patterns.

Challenge III: Scarcity of Critical Vulnerability Examples Many critical CWEs, particularly among the MITRE Top 25 Most Dangerous CWEs [17], are underrepresented in existing datasets. This severe imbalance limits the effectiveness of trained models in identifying rare but high-risk vulnerabilities.

Summary of Solutions To address these challenges, this paper introduces several solutions, including BenchVul for benchmarking and TitanVul for training vulnerability detection models.

Our Solution: BenchVul To address *Challenge I*, we introduce BenchVul, a manually curated benchmark focused on the MITRE Top 25 Most Dangerous CWEs [17] (hereafter, "Top 25 CWEs"). To construct it, we first aggregated seven publicly available datasets, performed comprehensive intra- and cross-dataset deduplication, and standardized CWE annotations based on updated NVD records. Due to the large volume of initial data, we applied an initial LLM-based filtering step to remove unrelated or non-security-related code changes. To ensure sufficient complexity, we aimed to curate a balanced benchmark with exactly 50 verified vulnerable samples per CWE category for the Top 25 CWEs. This is important because underrepresented vulnerabilities do not indicate lower danger levels, such as Hard-Coded Credentials (CWE-798) or Command Injection (CWE-77), which appear infrequently (see Figure 3) but can have catastrophic consequences. In cases where real-world data was insufficient, we introduced the Realistic Vulnerability Generation (RVG) framework, which addresses *Challenge III*. The RVG framework utilizes a multi-agent LLM workflow that simulates realistic development and security audit processes: (1) *Context & Threat Modeler* designs practical attack scenarios; (2) *Vulnerable Implementer* creates corresponding self-contained vulnerable code; (3) *Security Auditor* identifies and remediates the vulnerability; and (4) *Security Reviewer* independently validates both the presence and correct remediation of the target CWE. This process ensures that the synthesized samples are both realistic and targeted to underrepresented but critical CWEs. Finally, to ensure the highest data quality, we conducted a manual analysis of all candidate samples. Following this initial review, we recruited seven researchers to further evaluate each sample against the following criteria: (1) it represents a genuine vulnerability, (2) it is self-contained at the function level, and (3) it is correctly labeled with the intended CWE. We refer to the proportion of samples meeting all three criteria as the benchmark's *correctness*. This validation process resulted in a *correctness* rate of 92%, yielding a high-quality, balanced, and self-contained benchmark covering the Top 25 CWEs.

Our Solution: TitanVul While BenchVul provides a high-quality and reliable evaluation resource, its size is insufficient for training robust machine learning models, motivating the need for a larger and high-quality training dataset. To solve *Challenge II*, we first aggregated and merged seven publicly available vulnerability datasets and conducted extensive deduplication. To ensure data quality at scale, we applied a novel multi-agent LLM-based framework that automatically analyzed and validated each vulnerability–fix pair. Specifically, this framework consists of independent agents acting as *Auditor*, *Critic*, and *Consensus*: the *Auditor* reviews the evidence for each fix, the *Critic* challenges and verifies the auditor's assessment, and the *Consensus* agent synthesizes these judgments to filter out noisy or irrelevant samples. To prevent data leakage and ensure evaluation integrity, we further removed any overlapping samples between BenchVul and TitanVul. Through this multi-stage process, the initial set of 305,692 candidate functions was reduced to a final dataset of 35,045 validated vulnerable functions with corresponding fixes suitable for training vulnerability detection models. Given the scarcity of critical vulnerabilities in real-world data, we further explore the feasibility of augmenting TitanVul with realistic synthesized vulnerabilities generated by the RVG framework, to assess whether this augmentation improves model performance. Notably, there is no duplication between the synthesized vulnerabilities used for augmentation and those in BenchVul, ensuring evaluation integrity.

Evaluation To assess model generalization, we trained state-of-the-art models on a range of public datasets (including TitanVul) and evaluated their performance on our manually verified BenchVul benchmark. Notably, there is no data overlap between BenchVul and any training datasets, ensuring the integrity of the evaluation. Our results reveal a substantial *generalization gap*: models often achieve high self-testing accuracy when trained and tested on the same dataset but experience severe performance drops when evaluated on BenchVul. For example, UniXcoder achieves a self-testing accuracy of 0.776 on BigVul, but this falls by 33% to 0.519 on BenchVul. Comparable declines are observed with other datasets, such as CVEfixes (0.713 to 0.607), PrimeVul (0.567 to 0.337), and DiverseVul (0.641 to 0.402), indicating widespread overfitting to dataset-specific artifacts.

In contrast, models trained on our high-quality TitanVul dataset demonstrate superior generalization performance, achieving an accuracy of 0.767 on BenchVul despite a modest self-testing accuracy of 0.584. This improvement is attributable to the multi-agent LLM validation framework used in constructing TitanVul, which ensures that each vulnerability–fix pair is both genuine and self-contained at the function level. Moreover, augmenting TitanVul with realistic synthesized data from our RVG framework further enhances performance, increasing accuracy to 0.874 (a 14.0% improvement).

Main Contributions Our main contributions are:

- **BenchVul**, the first comprehensive, manually verified benchmark covering the Top 25 CWEs, with 50 vulnerable functions and their corresponding fixes per weakness (100 samples per weakness), yielding a total of over 1,000 verified vulnerable functions.
- **TitanVul**, a large-scale (35,045 functions), high-quality training dataset curated from seven public sources using rigorous deduplication and a novel multi-agent LLM verification framework to ensure high quality.
- Realistic Vulnerability Generation (RVG) framework, a multi-agent approach to synthesize realistic, context-aware data for underrepresented CWEs. We used RVG to generate synthetic

and realistic samples for BenchVul, specifically targeting Top 25 CWEs with insufficient representation.
- A large-scale empirical study that systematically quantifies intra- and cross-dataset duplication and CWE coverage across major public vulnerability datasets, revealing fundamental issues in current resources.

The remainder of this paper is organized as follows: Section 2 presents an empirical analysis of existing vulnerability datasets. Section 3 and Section 4 describe the construction of BenchVul and TitanVul. Section 5 and Section 6 show the experimental setup and results. Section 7 discusses key implications. Section 8 reviews related literature, and Section 9 concludes the paper.

2 Empirical Study

In this section, we study the characteristics of seven publicly available function-level vulnerability datasets that we selected for analysis: BigVul [7], CleanVul [15], CVEfixes [1], DiverseVul [3], PrimeVul [5], SafeCoder [14], and VulnPatchPairs [22]. We specifically focus on the intra-dataset duplications, cross-dataset duplication, and distributions of CWE types across these datasets. This analysis helps us understand the limitations of the publicly available vulnerability datasets and provides the foundation for our unified benchmark construction.

Table 1: Vulnerability Deduplication Analysis.

Dataset	Complete Pair Duplication			Self-Identical Duplication			Cross-Matched Conflict		
	Initial	After	Removed (%)	Remain	After	Removed (%)	Remain	After	Removed (%)
BigVul	188,635	188,474	161 (0.08%)	188,474	10,632	177,842 (94.36%)	10,632	10,281	351 (3.30%)
CleanVul	43,029	43,029	0 (0.00%)	43,029	43,029	0 (0.00%)	43,029	43,029	0 (0.00%)
CVEfixes	41,829	19,247	22,582 (53.99%)	19,247	17,743	1,504 (7.81%)	17,743	17,249	494 (2.78%)
DiverseVul	14,484	14,476	8 (0.06%)	14,476	13,974	502 (3.47%)	13,974	13,964	10 (0.07%)
PrimeVul	4,704	4,704	0 (0.00%)	4,704	4,704	0 (0.00%)	4,704	4,704	0 (0.00%)
SafeCoder	1,268	1,252	16 (1.26%)	1,252	1,242	10 (0.80%)	1,242	1,241	1 (0.08%)
VulnPatchPairs	11,743	11,743	0 (0.00%)	11,743	11,377	366 (3.12%)	11,377	11,260	117 (1.03%)
Total	305,692	282,925	22,767 (7.45%)	282,925	102,701	180,224 (63.70%)	102,701	101,728	973 (0.95%)

2.1 Intra-Dataset Duplications

Duplication presents a significant challenge in vulnerability datasets, potentially skewing analysis results and model performance [5]. We examined duplication rates across the seven datasets, identifying three distinct types: 1) Complete pair duplication (entire vulnerable-fixed code pairs appearing multiple times), 2) Self-identical duplication (vulnerable code identical to its fixed version), and 3) Cross-matched conflict (vulnerable code identical to fixed code from different pairs). For duplication detection, code normalization was performed by removing all whitespace characters (spaces, tabs, newlines, etc.). The results are presented in Table 1.

Widespread Duplication Across Datasets Our analysis reveals that duplication is a widespread issue across publicly available vulnerability datasets. In total, 22,767 redundant pairs (7.45%) were removed, with the highest duplication rate observed in CVEfixes (53.99%). The second filtering stage eliminated 180,224 self-identical pairs (63.70% of the remaining corpus), primarily from BigVul (94.36% of its pairs). A third step identified and removed 973 cross-matching conflicts (0.95%). After these three rounds of duplication removal, the number of vulnerable–fixed pairs was reduced from 305,692 to 102,701, a reduction of 66.4%. These results indicate that intra-dataset duplication is both widespread and unevenly distributed, underscoring the need for rigorous data validation in vulnerability research.

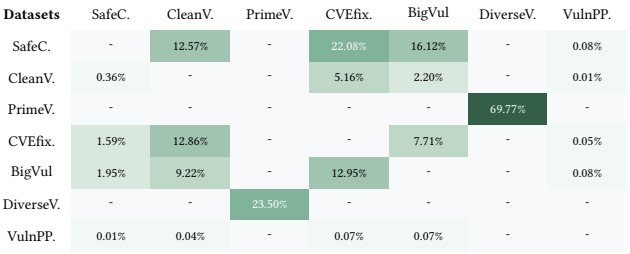


Figure 1: Vulnerability Duplication Matrix Across Datasets.

2.2 Cross-Dataset Duplications

In addition to intra-dataset duplication, cross-dataset duplication can impact the validity and uniqueness of datasets. Overlapping samples between different datasets can artificially inflate evaluation results and reduce the generalizability of vulnerability detection models. As shown in Figure 1, duplication rates vary widely: PrimeVul and DiverseVul share 69.77% of samples, while CVEfixes overlaps with SafeCoder (22.68%) and CleanVul (12.86%). By contrast, many other pairs have less than 1% overlap, indicating some datasets remain largely distinct.

2.3 Distribution of CWE Types

Understanding the distribution of CWE types in each dataset is essential for both training and evaluation: if a model is trained or tested on a dataset skewed toward certain vulnerabilities, its ability to generalize to real-world threats or underrepresented CWEs will be limited. A clear view of dataset focus and coverage also guides effective benchmark design and model interpretation. Figure 2 presents the distribution of labeled CWE types across six major vulnerability datasets. VulnPatchPair is not included because it does not provide CWE information. Each subplot displays the frequency of each CWE, sorted in descending order. CWEs classified among the MITRE Top 25 [17] are highlighted in green to indicate their prevalence.

Significant Imbalances and Dataset-Specific Biases Our analysis reveals substantial imbalances across all datasets, with the top 5-10 CWE types typically comprising 55-80% of all samples. The frequency ratios between the most and least common CWEs range from 11:1 (DiverseVul) to 155:1 (CleanVul), indicating severe class imbalance that could bias model training and evaluation. More critically, each dataset exhibits distinct vulnerability type biases that reflect their origins and intended use cases. Memory-related vulnerabilities dominate BigVul and PrimeVul (CWE-119, CWE-125, CWE-787). Conversely, CleanVul and SafeCoder are heavily skewed toward web application vulnerabilities (CWE-79 and CWE-89). CVEfixes shows concentration in SQL injection vulnerabilities, while DiverseVul demonstrates the most balanced distribution among major CWE types. These dataset-specific biases create significant challenges for training and evaluating vulnerability detection models, as models trained on one dataset may fail to generalize effectively to vulnerabilities prevalent in others.

Challenges of Severe CWE Imbalance Given the varied and dataset-specific biases observed in individual vulnerability datasets, we merged all datasets to examine a broader, unified perspective

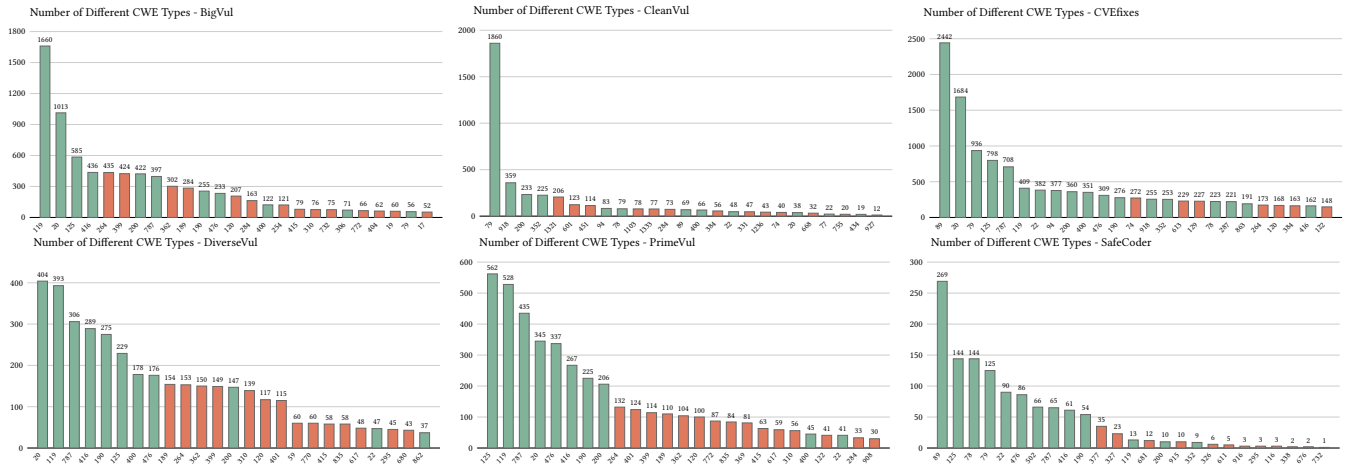


Figure 2: Distribution of CWE Types Across Six Major Vulnerability Datasets.

of CWE distribution. Figure 3 presents the resulting distribution of MITRE Top 25 most dangerous CWEs across this consolidated vulnerability dataset. Notably, there is substantial imbalance, with the frequency ratio between the most common CWE-20 and the least common CWE-798 reaching 166:1. This severe skew in data distribution underscores significant challenges for machine learning models in accurately detecting and generalizing across a wide spectrum of vulnerability types. Moreover, it is challenging to use these datasets as benchmarks, since many most dangerous CWE types are severely underrepresented. Addressing this imbalance through targeted dataset augmentation and careful benchmark design is therefore crucial for developing more comprehensive and robust vulnerability detection systems.

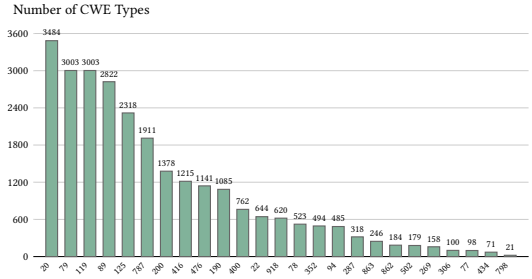


Figure 3: Distribution of MITRE Top 25 Most Dangerous CWE Across the Consolidated Vulnerability Dataset.

3 BenchVul: A Benchmark for the Top 25 Most Dangerous CWE Weaknesses

The construction of BenchVul, a comprehensive benchmark for evaluating vulnerability detection approaches across the MITRE Top 25 Most Dangerous CWEs [17], follows a multi-stage approach, as illustrated in Figure 4. We detail each stage of this process below.

3.1 Data Integration

We first aggregated multiple publicly available vulnerability datasets, including BigVul [7], CleanVul [15], CVEfixes [1], DiverseVul [3], PrimeVul [5], SafeCoder [14], and VulnPatchPairs [22]. To ensure the benchmark's diversity and reliability, we standardized these datasets into a unified format suitable for integrated analysis. Manual inspection of the aggregated datasets revealed inconsistencies in CWE labeling compared to the National Vulnerability Database (NVD), largely due to outdated or missing annotations. To address these discrepancies, we updated each vulnerability's CWE information by retrieving the latest annotations from the NVD based on CVE identifiers. We observed that initially, only 74.83% of PrimeVul, 43.01% of DiverseVul, and 70.97% of BigVul samples matched the NVD's CWE annotations. Across all datasets, 12,127 vulnerability instances had inconsistent CWE labels, all of which were corrected using updated NVD records. Additionally, datasets lacking CWE annotations, such as CVEfixes, were supplemented by deriving CWE identifiers directly from their corresponding CVE records. This normalization resulted in a consolidated dataset containing a total of 305,692 vulnerability-fix pairs. We then applied intra-dataset deduplication (Section 2.1), removing complete pair, self-identity, and cross-identity duplicates, reducing the set to 101,728 pairs. Next, we merged the cleaned datasets and performed cross-dataset deduplication (Section 2.2) to obtain a unified vulnerability dataset.

3.2 LLM-Based Filtering

To construct a high-quality, function-level benchmark covering the MITRE Top 25 Most Dangerous CWEs [17], each vulnerability-fixing pair must be manually verified. Specifically, we aim to ensure that each pair accurately represents a genuine vulnerability fix and is self-contained, meaning the vulnerability fix can be fully understood by examining only the code within a single function [23]. However, manually validating every pair is impractical due to the large volume of samples available for some CWEs (e.g., over 3,000 instances as shown in Figure 3). Furthermore, prior studies indicate that a significant portion of labeled vulnerabilities do not genuinely address security flaws but rather represent unrelated bug fixes, refactoring, or other code changes [3, 5]. To efficiently address these challenges, we first leverage LLMs to filter out unrelated code changes, substantially reducing the number of candidate samples

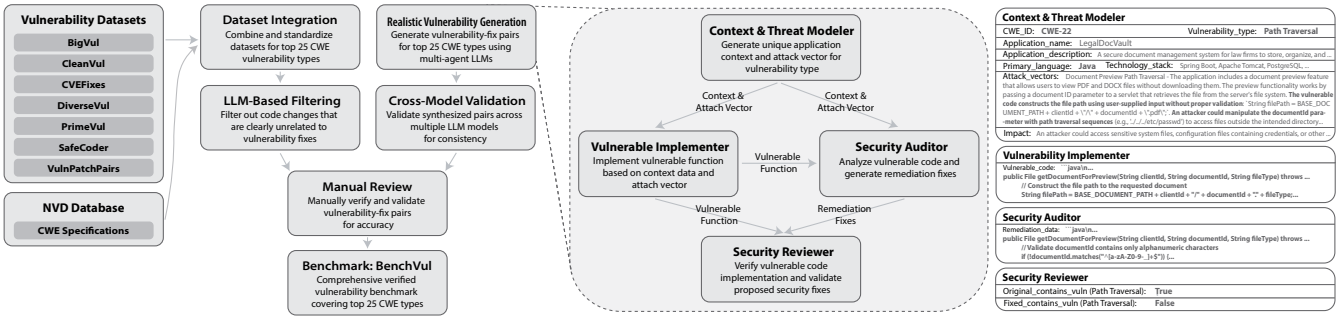


Figure 4: Overview of the BenchVul construction pipeline for the MITRE Top 25 Most Dangerous CWEs. Vulnerability data from seven public datasets and NVD CWE labels are integrated and deduplicated. LLM-based filtering and manual review ensure genuine, self-contained vulnerability-fix pairs. For underrepresented CWEs, the multi-agent RVG framework synthesizes realistic code pairs, which undergo cross-model validation. All benchmark samples receive human verification.

requiring manual verification [15]. Although LLM-based filtering can occasionally introduce false positives or negatives, we mitigate this risk through subsequent structured manual reviews (see Section 3.5), where each remaining sample is carefully checked to confirm it represents a genuine and self-contained vulnerability fix. This combined approach ensures the final benchmark maintains high accuracy while significantly improving validation efficiency.

3.3 Realistic Vulnerability Generation

To construct a robust benchmark with at least 50 vulnerable and 50 corresponding fixed functions per CWE type, sufficient real-world examples of self-contained vulnerabilities are necessary. However, some CWE categories lack adequate real-world examples, making it necessary to synthesize additional realistic vulnerability-fixing pairs. To address this challenge, we propose the **Realistic Vulnerability Generation** (RVG) framework, a multi-agent LLM approach illustrated in Figure 4. The RVG framework comprises four interrelated roles: *Context & Threat Modeler*, *Vulnerable Implementer*, *Security Auditor*, and *Security Reviewer*. Each role contributes to generating realistic, validated vulnerability pairs, detailed as follows. Due to space constraints, the full prompts and scripts are provided in the replication package[1].

Context & Threat Modeler Given a CWE ID and name as inputs, this agent initiates the RVG process by creating a realistic application context and identifying a corresponding attack vector. To maximize diversity and realism, this agent selects a distinct programming language, technology stack, user roles, and functionalities for each scenario. It also maintains uniqueness by tracking previously generated contexts, employing a first-in-first-out (FIFO) approach to prevent repetition.

Vulnerable Implementer This agent generates a realistic and self-contained vulnerable code snippet based explicitly on the context and attack vector defined by the previous agent. The code incorporates subtle but exploitable vulnerabilities, accompanied by comments describing the intended functionality without indicating vulnerabilities. This approach closely mimics real-world development scenarios.

Security Auditor The Security Auditor analyzes the vulnerable code snippet to identify security flaws and subsequently produces a secure, production-ready version of the same code. Each code modification is documented through comments, clearly indicating the original vulnerability and the rationale behind each remediation, following established security best practices.

Security Reviewer This agent performs a comparative evaluation of the vulnerable and remediated code snippets. It objectively verifies whether the identified CWE-related vulnerability is present in the vulnerable snippet and properly mitigated in the remediated snippet. The Security Reviewer provides a concise assessment of the effectiveness of the remediation.

3.4 Cross-Model Validation

To strengthen the robustness of synthesized vulnerability data, we conducted cross-model validation using different state-of-the-art LLMs. Specifically, we utilized Claude 3.7 Sonnet for initial synthesis tasks and GPT-4o for validation purposes. Each synthesized vulnerability-fix pair generated by Claude 3.7 was independently assessed by GPT-4o, verifying whether the vulnerability was correctly implemented and effectively remediated.

3.5 Manual Review

After the automated filtering and synthesis stages, all remaining vulnerability–fix pairs underwent a structured human review. We first manually verified that every pair (1) represents a genuine vulnerability, (2) is self-contained at the function level, and (3) is correctly labeled with the intended CWE. Where real-world data were insufficient, synthesized pairs produced by the RVG framework were reviewed with the same criteria. This procedure yielded exactly 50 validated vulnerable functions and 50 corresponding remediations for each of the MITRE Top 25 Most Dangerous CWEs [17], producing a final benchmark with over 1,000 vulnerability–fix pairs. To assess benchmark quality, seven independent researchers with experience in vulnerability analysis participated in the review.

[1]https://github.com/yikun-li/TitanVul-BenchVul

Each researcher was assigned a random sample of the benchmark and asked to evaluate whether the vulnerabilities met the aforementioned criteria. Of 275 reviewed pairs, 253 were judged correct, corresponding to an overall *correctness* rate of 92%. While the benchmark labels are not perfect, achieving a *correctness* rate above 90% is widely considered sufficient for reliable evaluation in empirical software engineering research [21], providing confidence that our benchmark effectively supports rigorous vulnerability detection studies. These results demonstrate that the curated benchmark provides both high label accuracy and comprehensive coverage of the Top 25 CWE weaknesses.

4 TitanVul: A Large-Scale and High-Quality Vulnerability Dataset

Vulnerability detection models require not only evaluation benchmarks, but also large, high-quality training datasets for training. While BenchVul offers manually verified data for evaluation, its limited scale and the cost of manual validation make it impractical for training machine learning models. Thus, scalable methods are needed to curate function-level vulnerability data for effective model training. Existing vulnerability datasets vary widely in quality. Prior studies report that only a fraction of samples in several popular datasets represent valid vulnerability fixes, where validity is defined as correctly identifying all code changes associated with a vulnerability fix (e.g., only the relevant changes, without unrelated modifications) [3, 5]: *BigVul* (25.0%), *VulnPatchPairs* (36.0%), *CVEfixes* (51.7%), and *DiverseVul* (60.0%). In contrast, *CleanVul* and *PrimeVul* achieve higher validity rates of 90.6% and 86.0%, respectively. However, high validity alone does not ensure that the vulnerability and its fix can be clearly understood (self-contained) from the CWE label, commit message, and code diff. If this context is missing, such samples are unlikely to help models learn true vulnerability detection at the function level. While BenchVul provides a high-quality, manually verified set of over 1,000 vulnerability–fix pairs, its limited size and the resource-intensive nature of manual validation make it impractical to use as a large-scale training dataset. To develop robust and generalizable vulnerability detection models, there is a clear need for larger, high-quality datasets that are both reliable and scalable. To address this challenge, we re-examined the consolidated dataset (Section 2), removing noise and excluding vulnerability–fix pairs that were not self-contained. We automated this process using a multi-agent framework leveraging LLMs for comprehensive analysis, verification, and validation of security vulnerabilities. The architecture of the framework is illustrated in Figure 8, comprising three key components: *Vulnerability Auditor*, *Vulnerability Critic*, and *Vulnerability Consensus*.

Vulnerability Auditor This agent serves as the initial evaluator, analyzing code diffs, commit messages, and associated CWE information. Its primary role is to determine whether the submitted changes represent genuine security vulnerability fixes. The Auditor provides detailed evidence by identifying the type of vulnerability addressed, highlighting relevant code snippets, and incorporating insights from commit messages or CWE hints. This agent ensures that its assessments are grounded in concrete observations from the codebase.

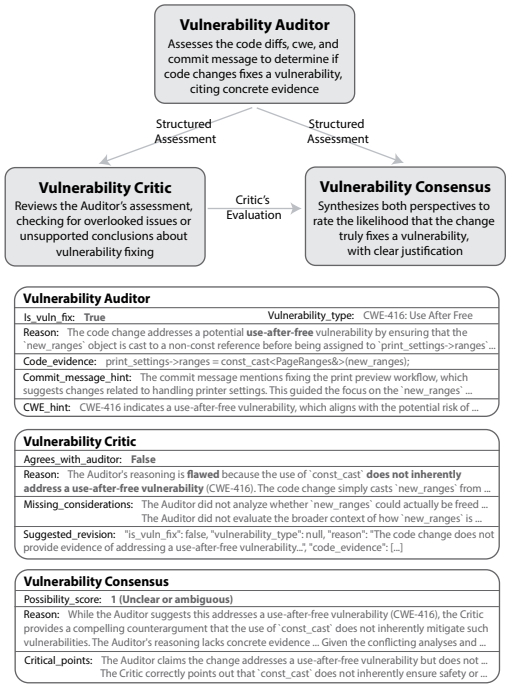


Figure 5: Overview of the multi-agent LLM verification framework used to construct TitanVul, a large-scale, high-quality vulnerability dataset. The *Vulnerability Auditor* initially assesses code diffs, commit messages, and CWE labels to identify vulnerability fixes, providing structured evidence. The *Vulnerability Critic* reviews the Auditor's assessment, identifying overlooked issues or inaccuracies. Finally, the *Vulnerability Consensus* synthesizes these perspectives and assigns a possibility score reflecting the likelihood that the code change fixes a security vulnerability.

Vulnerability Critic This agent performs a secondary review, scrutinizing the Auditor's findings for accuracy, completeness, and robustness. It identifies any overlooked issues, incorrect reasoning, or weak evidence in the Auditor's analysis. By providing constructive feedback and corrections, the Critic ensures a thorough and reliable evaluation of each vulnerability fix.

Vulnerability Consensus This agent synthesizes the analyses from the Auditor and Critic to produce a unified and justified assessment. It assigns a possibility score (ranging from 0 to 3) indicating the likelihood that the code change genuinely addresses a security vulnerability. This consensus-building process carefully considers both agreement and disagreement points among previous analyses, prioritizing concrete evidence and clearly articulating its reasoning.

TitanVul We begin by performing comprehensive deduplication and merging of datasets, updating CWE labels. Next, we employ our multi-agent framework to further enhance data quality, ensuring that each security vulnerability fix included in TitanVul is rigorously validated and accurately represented. To prevent any potential data leakage, we then remove duplicate samples between

BenchVul and other sources in the finalized dataset. The resulting dataset comprises 35,045 vulnerable functions along with their corresponding fixes, establishing TitanVul as a reliable resource for vulnerability-related research and applications. Due to space limitations, the detailed prompts and scripts used in our framework are provided in the replication package[1].

5 Experimental Setup

5.1 Research Questions

We formulate the following research questions (RQs):

RQ1: How well can models trained on vulnerability datasets detect the Top 25 Most Dangerous CWEs? Due to the lack of vulnerability benchmarks, most prior studies evaluate models on the same datasets used for training, making it difficult to assess true generalization [23]. Given the widespread issues of overfitting and dataset bias, it is critical to rigorously evaluate whether models can actually identify the Top 25 CWE weaknesses on an independent and high-quality benchmark (BenchVul).

RQ2: How does the choice of training dataset affect model performance across CWE categories? Our analysis reveals that publicly available datasets differ widely in their CWE distribution and quality, with each exhibiting distinct biases toward certain vulnerability types (e.g., memory safety, web security). Understanding how these differences impact model performance can illuminate the strengths and weaknesses of popular datasets and inform future dataset construction and model development.

RQ3: Does adding synthesized data improve detection of the Top 25 Most Dangerous CWEs? Since many of the most dangerous CWEs are rare in real-world datasets, models may lack sufficient examples to learn robust patterns. Synthetic data generation offers a potential solution by augmenting scarce categories and improving model coverage. Evaluating the actual benefit of synthesized data for detecting critical weaknesses is thus important for advancing practical ML-based vulnerability detection.

5.2 Models

We evaluate a diverse set of state-of-the-art language models for vulnerability detection, including encoder-only models (CodeBERT [8], GraphCodeBERT [12]), the unified encoder–decoder model UniXcoder [11], and decoder-only models (GPT-2 [20], Llama-3.2-3B [10], DeepSeek-Coder-1.3B [13]). This selection enables a comprehensive comparison of architectural styles and model scales in the context of vulnerability detection.

5.3 Evaluation Metrics

We evaluate model performance using four standard metrics: **accuracy**, **precision**, **recall**, and **F1-score**. Accuracy reflects the proportion of correctly classified samples in our balanced dataset. Precision and recall measure, respectively, how many predicted vulnerabilities are correct and how many actual vulnerabilities are detected. The F1-score, the harmonic mean of precision and recall, provides an overall balance between these two metrics.

5.4 Implementation Details

For training, the dataset is split into training (70%), validation (15%), and test (15%) sets using random stratified sampling. Training is conducted for up to 50 epochs, and the best-performing checkpoints are retained for evaluation. All experiments are run on NVIDIA H100 GPUs with an Intel Xeon Platinum 8480C CPU.

6 Results

6.1 RQ1: Dataset Performance on Top 25 CWEs

We evaluated the effectiveness of language models trained on various vulnerability datasets in detecting the MITRE Top 25 Most Dangerous CWEs [17] using our curated benchmark, BenchVul. Table 2 presents comprehensive results across eight datasets and six model architectures, revealing critical insights about generalization capabilities in vulnerability detection. We primarily use accuracy as the evaluation metric because BenchVul contains a balanced number of vulnerable and non-vulnerable samples for each CWE, making accuracy straightforward to interpret and directly comparable to the random guessing baseline (0.5). In contrast, metrics such as F1-score can sometimes be misleading. For instance, a naive model that predicts all samples as vulnerable would achieve perfect recall (1.0), precision of 0.5, and thus an inflated F1-score of 0.667, despite performing no better than random guessing.

The Generalization Gap: Self-Testing vs. Cross-Dataset Performance Our analysis reveals a pattern of poor generalization across existing vulnerability datasets. UniXcoder, which consistently achieves the strongest performance across datasets, exemplifies this trend. When trained on BigVul, it achieves an impressive self-testing accuracy of 0.776 but suffers a 33% performance drop to 0.519 on BenchVul — barely above random chance. This pattern persists across popular datasets: CVEfixes (0.713 → 0.607), PrimeVul (0.567 → 0.337), and most severely, DiverseVul (0.641 → 0.402). The magnitude of these performance gaps suggests that existing datasets may promote memorization rather than genuine vulnerability pattern recognition. Models appear to overfit to dataset-specific characteristics, failing to capture generalizable vulnerability signatures.

TitanVul: Prioritizing Generalization Over Self-Testing Performance Our TitanVul demonstrates a fundamentally different trade-off between self-testing and generalization performance. While UniXcoder achieves a modest self-testing accuracy of 0.584, lower than most competing datasets, it delivers exceptional cross-dataset performance of 0.767 on BenchVul, representing the highest generalization capability across all evaluated datasets. This performance inversion is particularly evident when comparing TitanVul to datasets like BigVul and DiverseVul, which achieve high self-testing scores but fail catastrophically on independent evaluation. The pattern holds across multiple models: DeepSeek-Coder-1.3B trained on TitanVul achieves 0.788 cross-dataset accuracy, substantially outperforming its 0.583 self-testing performance.

Consistency of Findings on C/C++ Specific Benchmarking To further validate our findings in a controlled setting, we conducted additional experiments focusing exclusively on C/C++ samples from BenchVul, recognizing that some vulnerability datasets

Table 2: Performance Comparison of Language Models Trained on Different Vulnerability Datasets and Tested on BenchVul.

Model	Trained on BigVul					Trained on CVEfixes					Trained on CleanVul					Trained on DiverseVul				
	Self	Acc	Pre	Rec	F1	Self	Acc	Pre	Rec	F1	Self	Acc	Pre	Rec	F1	Self	Acc	Pre	Rec	F1
CodeBERT	0.691	0.530	0.518	0.886	0.650	0.616	0.552	0.538	0.427	0.397	0.561	0.566	0.561	0.859	0.663	0.500	**0.500**	0.500	1.000	**0.667**
GraphCodeBERT	0.699	**0.576**	0.559	0.767	0.642	0.599	0.555	0.553	0.837	0.650	0.615	0.664	0.677	0.676	0.665	0.549	0.394	0.387	0.371	0.364
GPT2	0.661	0.521	0.518	0.657	0.571	0.663	0.542	0.539	0.605	0.568	0.556	0.538	0.537	0.608	0.568	0.531	0.498	0.499	0.508	0.500
UniXcoder	**0.776**	0.519	0.511	0.912	**0.651**	**0.713**	**0.607**	0.589	0.743	**0.652**	**0.665**	**0.695**	0.681	0.780	0.719	**0.641**	0.402	0.309	0.137	0.184
DeepSeek-Coder-1.3B	0.744	0.516	0.510	0.906	0.650	0.670	0.555	0.540	0.788	0.639	0.638	0.684	0.643	0.897	**0.745**	0.588	0.403	0.369	0.284	0.315
Llama-3.2-3B	0.735	0.514	0.508	0.914	**0.651**	0.666	0.473	0.474	0.486	0.477	0.605	0.683	0.656	0.824	0.726	0.565	0.574	0.557	0.735	0.632

Model	Trained on TitanVul					Trained on PrimeVul					Trained on SafeCoder					Trained on VulnPatchPair				
	Self	Acc	Pre	Rec	F1	Self	Acc	Pre	Rec	F1	Self	Acc	Pre	Rec	F1	Self	Acc	Pre	Rec	F1
CodeBERT	0.513	0.504	0.610	0.024	0.046	0.524	0.409	0.384	0.381	0.343	0.637	0.604	0.597	0.761	0.655	0.503	0.507	0.489	0.586	0.432
GraphCodeBERT	0.515	0.560	0.600	0.420	0.494	0.517	0.424	0.426	0.676	**0.504**	0.666	0.648	0.650	0.656	0.648	0.546	**0.585**	0.584	0.700	0.626
GPT2	0.538	0.600	0.752	0.297	0.426	0.519	0.452	0.453	0.435	0.439	0.582	0.553	0.560	0.396	0.454	0.538	0.507	0.506	0.592	0.540
UniXcoder	**0.584**	0.767	0.805	0.706	0.752	**0.567**	0.337	0.290	0.186	0.217	**0.713**	**0.716**	0.730	0.736	0.723	**0.595**	0.523	0.516	0.878	**0.646**
DeepSeek-Coder-1.3B	0.583	**0.788**	0.826	0.729	**0.774**	0.531	0.393	0.349	0.225	0.266	0.681	0.671	0.658	0.739	0.688	0.589	0.542	0.546	0.544	0.526
Llama-3.2-3B	0.583	0.654	0.682	0.576	0.625	0.508	**0.523**	0.542	0.370	0.421	0.664	0.683	0.656	0.824	**0.726**	0.550	0.484	0.483	0.545	0.497

Note: "Self" refers to the accuracy of training and testing on the same dataset, while other metrics (Acc, Pre, Rec, F1) represent performance when training on the respective dataset and testing on BenchVul. The highest accuracy and F1-score values in each column are shown in **bold** and highlighted in dark green. Other accuracy values above 0.6 and F1-scores above 0.7 are highlighted in green. Accuracy values below 0.4 and F1-scores below 0.3 are highlighted in red.

(BigVul, DiverseVul, PrimeVul, and VulnPatchPair) are predominantly composed of C/C++ code. Table 3 presents the performance of UniXcoder trained on different datasets when evaluated specifically on 230 vulnerable C/C++ pair samples from BenchVul. The results confirm our observations from the multi-lingual evaluation, demonstrating that the generalization challenges persist even when restricting evaluation to the same programming language domain as the training data. Notably, TitanVul continues to demonstrate superior cross-dataset performance (0.730 accuracy) compared to traditional C/C++ focused datasets like BigVul (0.552) and DiverseVul (0.387). The consistency between these C/C++ specific results and our broader multi-lingual findings reinforces that existing datasets' limitations stem from fundamental issues in data quality rather than language-specific factors.

Table 3: Performance of UniXcoder Trained on Different Datasets and Evaluated on C/C++ Samples from BenchVul.

Dataset	Self	Acc	Pre	Rec	F1
BigVul	**0.776**	0.552	0.528	0.987	0.687
CVEfixes	0.713	0.621	0.619	0.635	0.627
CleanVul	0.665	0.674	0.660	0.717	**0.688**
DiverseVul	0.641	0.387	0.309	0.183	0.230
TitanVul	0.584	**0.730**	0.853	0.557	0.674
PrimeVul	0.567	0.343	0.329	0.300	0.314
SafeCoder	0.713	0.676	0.671	0.691	0.681
VulnPatchPair	0.595	0.517	0.514	0.648	0.573

Model Complexity and Dataset Quality Interactions The relationship between model size and performance reveals important insights into dataset quality. Across most existing datasets, the smaller UniXcoder model consistently outperforms the larger DeepSeek-Coder-1.3B. For instance, on CleanVul and SafeCoder, UniXcoder achieves 0.695 and 0.716 cross-dataset accuracy, while the larger model only reaches 0.684 and 0.671. It suggests that existing datasets may lack the complexity or quality necessary to benefit from larger model capacity, or that their inherent biases can be exploited equally well by smaller, more efficient architectures.

To further disentangle the effects of dataset size versus quality, we trained UniXcoder on the combined, deduplicated union of all public datasets. This experiment in maximizing data quantity over quality yielded a model with a self-testing accuracy of 0.556 and a generalization accuracy of 0.647 on BenchVul (F1-score of 0.673). This result is lower than the 0.767 accuracy achieved by training on our curated TitanVul, demonstrating that sheer volume of mixed-quality data cannot replace high-quality data.

TitanVul, with its rigorous multi-agent verification, provides a clear counter-example where data quality allows larger models to excel. DeepSeek-Coder-1.3B achieves good performance on TitanVul (0.788 cross-dataset accuracy), outperforming UniXcoder (0.767). Notably, UniXcoder's relatively modest self-testing performance (0.584) on TitanVul suggests the model struggles to fully capture the dataset's complex patterns during training, yet still achieves strong generalization. This indicates that TitanVul's quality enables even partially-fitted models to learn robust vulnerability representations. In contrast, larger models appear to require high-quality, complex datasets to justify their computational overhead. They do not provide benefits when trained on smaller or lower-quality datasets that can be adequately handled by more efficient architectures.

RQ1: ***High self-testing accuracy does not translate to reliable generalization*** *on independent benchmarks. Results show that* ***most existing datasets lead to overfitting rather than true vulnerability detection****. For example, UniXcoder's accuracy drops by 33% for BigVul, 15% for CVEfixes, 41% for PrimeVul, and 37% for DiverseVul when evaluated on BenchVul. In contrast,* ***training on TitanVul increases accuracy by 31%*** *(0.584 to 0.767), highlighting the effectiveness our approach in improve dataset quality.*

6.2 RQ2: Effect of Training Data Across CWEs

We conduct a detailed comparison of model performance across individual CWE categories to assess the impact of training datasets.

This analysis demonstrates that the selection of training dataset substantially influences generalization performance on a per-CWE basis. Results indicate that datasets can be stratified into high-, moderate-, and low-performing groups according to their performance on BenchVul, as shown in Figure 6 and Figure 7.

High-Performing Datasets: Achieving Robust and Generalizable Detection The top tier includes TitanVul, SafeCoder, and CleanVul, which produce models with high cross-dataset accuracies and consistently strong performance across the CWE spectrum. TitanVul leads with a 0.76 accuracy, and its per-CWE scores range from 0.58 to 0.93, indicating robust performance across all evaluated categories. SafeCoder (0.71 accuracy) and CleanVul (0.69 accuracy) follow, with tight performance ranges of 0.54-0.89 and 0.53-0.81, respectively. They excel across both memory and web-related flaws; for example, accuracies for the critical Use CWE-416 reach 0.83 on SafeCoder and 0.81 on CleanVul. This robust performance is attributable to their advanced curation methodologies, such as LLM-based filtering and multi-agent verification, which minimize noise and foster the learning of generalizable security patterns.

Moderately Performing Datasets: Consistent but Limited Generalization In a secondary tier, datasets such as CVEfixes, VulnPatchPairs, and BigVul produce models that fail to generalize effectively, with performance that is unreliable and, in some cases, only marginally better than random guessing. CVEfixes leads this underperforming group with a 0.61 accuracy. While it shows isolated peaks of performance on certain flaws like CWE-22 (0.73), its

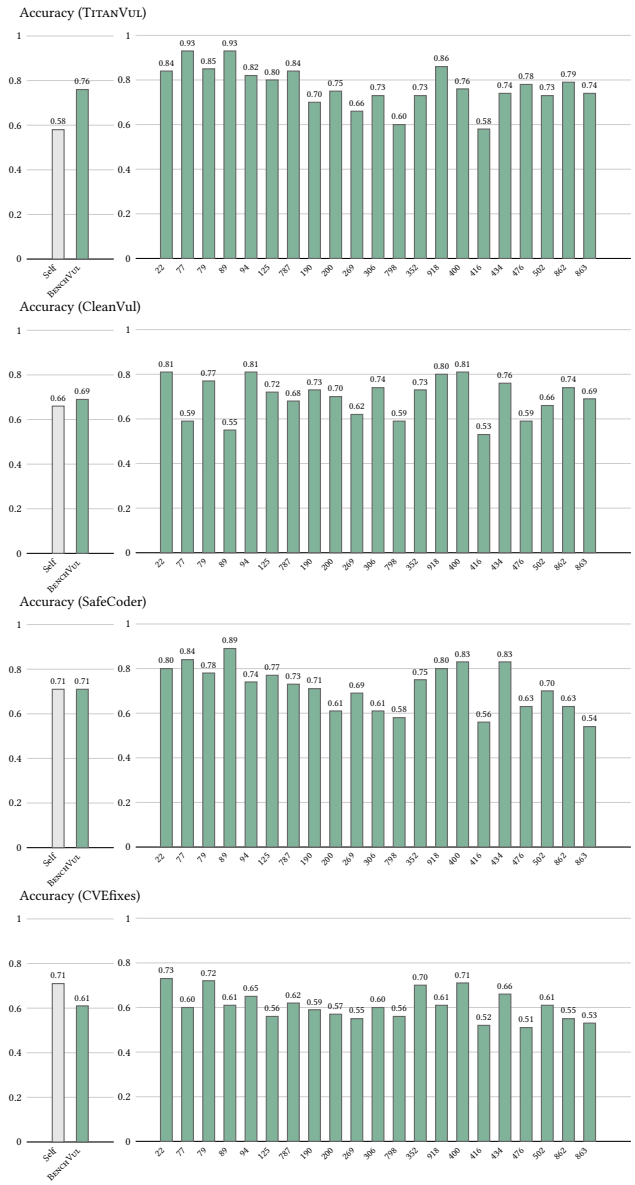


Figure 6: Performance of UnixCoder Trained on Different Datasets and Tested on BenchVul for Different CWE Types.

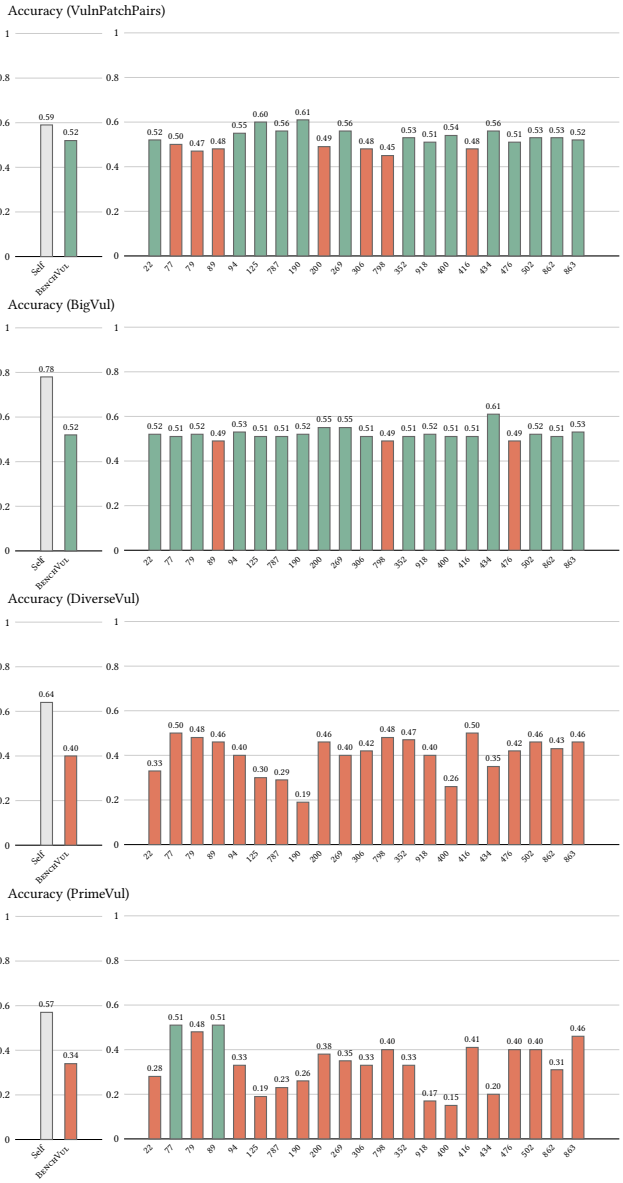


Figure 7: Performance of UnixCoder Trained on Different Datasets and Tested on BenchVul for Different CWE Types.

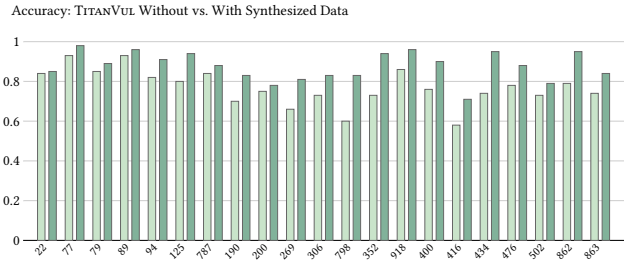
accuracy drops to 0.51 on others (CWE-502), demonstrating inconsistent and unreliable detection capabilities. The problem is more pronounced for VulnPatchPairs and BigVul, which both yield models with an overall accuracy of just 0.52, indicating a near failure to learn generalizable vulnerability patterns. These datasets exhibit a narrow, niche focus; for example, VulnPatchPairs performs slightly better on memory errors like CWE-190 (0.61), while BigVul's peak is on CWE-434 (0.61). However, this apparent specialization is a critical flaw, leading models to overfit on specific patterns while failing completely on others. This is evident in their inability to detect web flaws like CWE-89, where accuracies fall to 0.49. The domain-specific biases and lack of rigorous noise reduction in these datasets make them unsuitable for training broadly effective vulnerability detection models.

Poorly Performing Datasets: Limited Generalization and Overfitting At the lowest tier, DiverseVul and PrimeVul produce models that fail to generalize, reflected in low cross-dataset accuracies of 0.40 and 0.34. Their performance ranges are wide and dip to extremely low values: from a maximum of 0.50 down to 0.19 for DiverseVul, and from 0.51 down to 0.15 for PrimeVul. This failure is particularly severe on critical memory errors like CWE-190 and CWE-416. These results highlight how fundamental dataset limitations, such as shallow coverage (DiverseVul) or a combination of limited scale, and potentially excessive vulnerability complexity for function-level analysis (PrimeVul), which can lead to severe model overfitting.

RQ2: ***The choice of training dataset shapes model performance across CWE categories****. Models trained on* TitanVul *achieve robust generalization, with per-CWE ranging from 0.58 to 0.93. In contrast, models trained on datasets, such as PrimeVul and DiverseVul, struggle to surpass random guessing, averaging 0.34 and 0.40, and their highest per-CWE scores barely reach 0.51.* ***This indicates that data quality is more important than dataset size or CWE coverage for reliable vulnerability detection.***

6.3 RQ3: Impact of Synthesizing Training Data

Synthesizing vulnerability examples with LLMs offers a promising way to augment real-world vulnerability datasets, potentially addressing data scarcity for underrepresented CWEs. Vulnerabilities synthesized using various LLMs could improve different datasets, helping models achieve stronger generalization across multiple vulnerability types. To initially explore this hypothesis, we augmented the TitanVul training set (containing only real-world data) with synthesized vulnerabilities. Specifically, we added 100 new vulnerable samples for each of the Top 25 CWE types using our RVG approach (Section 3.3). It is noted that there is no duplication between the synthesized data and BenchVul, ensuring evaluation integrity. We then compared the performance of UniXcoder trained on the original TitanVul versus this augmented dataset. As shown in Figure 8, the inclusion of synthesized data significantly improves performance. The model's accuracy improves from 0.767 to 0.874 (a 14.0% increase), and its F1-score rises from 0.752 to 0.876 (a 16.5% increase), with consistent gains across nearly all individual CWE categories.



Dataset	Acc	Pre	Rec	F1
TitanVul w/o. Synthesized Data	0.767	0.805	0.706	0.752
TitanVul w. Synthesized Data	**0.874** (14.0%↑)	0.861 (7.0%↑)	0.892 (26.3%↑)	**0.876** (16.5%↑)

Figure 8: UnixCoder Performance on TitanVul w/o. Synthesized Data vs. TitanVul w. Synthesized Data.

Targeted Improvement for Under-Representative Weaknesses The benefits of data synthesis are most pronounced for weaknesses that are rare in the original dataset. A clear example is CWE-798 (Hard-Coded Credentials), for which only 21 samples existed in our consolidated dataset (Figure 3). This data scarcity limited the baseline model to just 0.60 accuracy; however, after augmentation, its accuracy surged to 0.83. Conversely, for vulnerabilities where the baseline model was already highly proficient, the gains were more modest. For instance, CWE-94 (Code Injection), with a high baseline accuracy of 0.96, saw only a minor increase. This demonstrates that data synthesis is a powerful tool for compensating for data scarcity while still offering incremental benefits for well-represented classes.

RQ3: Augmenting training data with synthesized data increases overall model accuracy by 14% *(0.767 to 0.874) across the Top 25 CWEs.* ***Gains are especially notable for underrepresented weaknesses****, such as CWE-798, where accuracy rises from 0.60 to 0.83 (a 38% increase).*

7 Discussion

The Deception of Self-Testing Our results challenge the validity of self-testing as a meaningful performance metric. We observed a significant generalization gap where models achieve high self-test scores but fail on independent evaluation. For instance, UniXcoder trained on BigVul sees its accuracy drop from 0.776 to 0.519 when moving from self-testing to BenchVul, indicating severe overfitting to dataset-specific artifacts. Notably, while UniXcoder achieves nearly identical self-testing accuracy on both TitanVul and PrimeVul (0.584 and 0.567, respectively), their generalization performance on BenchVul diverges notably: TitanVul yields a top accuracy of 0.767, while PrimeVul plummets to 0.337, more than a twofold difference. This result demonstrates that high self-testing scores do not refer to real generalizability, and underscores the necessity of standardized benchmarks to reliably assess a model's true vulnerability detection capabilities.

Rethinking Dataset Quality Beyond Validity Validity in vulnerability datasets is typically defined by accurately identifying all code changes associated with a vulnerability fix (e.g., three specific code changes in a commit, with no unrelated modifications). While this ensures that the labeled fix is technically correct, it does

not guarantee that the vulnerability fix is understandable from the available information. Our results show that datasets with high reported validity, such as PrimeVul (86.0%), can still lead to poor generalization (0.34 accuracy), while those with lower validity, like CVEfixes (51.7%), may perform better (0.61 accuracy) [3, 5]. If human experts cannot understand the nature of the vulnerability and its fix from the CWE label, commit message, and code diff alone, such samples are unlikely to be useful for training models to detect vulnerabilities at the function level. Therefore, for function-level vulnerability detection, it is essential that vulnerability–fix pairs are understandable (i.e., self-contained). In constructing TitanVul, we addressed this by excluding all pairs that did not meet this criterion.

Synergy of Model Complexity and Data Quality The benefits of complex models are contingent on the quality and complexity of the training data. On most existing datasets, the simpler UniXcoder model performed on par with or better than larger models like DeepSeek-Coder-1.3B. The larger model's advantage only emerged when trained on our higher-quality TitanVul. This highlights a synergistic relationship: advanced models only realize their potential when trained on large, high-quality datasets that are sufficiently complex to demand sophisticated learning.

Threats to Validity We acknowledge several potential validity concerns and outline mitigation steps to ensure these threats remain limited or comparable to those in prior studies. First, the validity of our BenchVul is a key consideration. To address this, we implemented a multi-stage validation process that included both automated filtering and manual review. Our additional manual assessment confirms a high *correctness* rate of 92%, demonstrating that BenchVul is accurate for evaluating vulnerability detection models. This approach to dataset validation is consistent with standards adopted in related empirical software engineering studies [21]. Second, our function-level detection focus is widely accepted in vulnerability detection research [16, 23]; both BenchVul and TitanVul have been specifically constructed and validated for function-level granularity, clearly delimiting the scope and applicability of our findings. Taken together, these mitigation strategies ensure the validity risks are effectively managed and comparable with best practices in the field.

8 Related Work

Vulnerability Datasets Early large-scale datasets like BigVul [7], created by mining vulnerability-fixing commits, suffer from significant noise, with correctness rates as low as 25.0% [5]. Subsequent datasets aimed to improve quality but introduced other limitations. For instance, CVEfixes [1] offers precise mapping to CVEs but has limited scope, while PrimeVul [5] achieves 86.0% correctness by focusing on single-function commits, potentially sacrificing realism. Similarly, DiverseVul [3] prioritizes language diversity at the cost of correctness (60.0%). A significant advancement came with CleanVul [15], which pioneered the use of LLMs to filter noisy commits, achieving 90.6% correctness and establishing a new standard for data quality. Our work builds on these efforts. We construct TitanVul by applying a rigorous, multi-agent LLM verification framework to a large, aggregated corpus, resulting in a dataset that uniquely combines both large scale and high quality. Moreover, there is a need for an independent benchmark to objectively assess model generalization, rather than relying on self-evaluation, which tend to overestimate real-world performance [23]. Thus, we introduce BenchVul, the first manually-verified benchmark to provide balanced and comprehensive coverage of the MITRE Top 25 Most Dangerous CWEs [17]. Together, these contributions enable more reliable evaluation and foster the development of models with true real-world generalizability.

Vulnerability Synthesis To augment datasets, especially for underrepresented weaknesses, researchers have explored automated vulnerability synthesis. One established approach is programmatic injection, exemplified by LAVA [6], which mutates benign code to deterministically insert flaws, but these tend to lack realism. More recent work [18, 19], applies neural code editing models to generate vulnerabilities by learning edit operations from vulnerable/fixed code pairs, producing more authentic flaws than rule-based mutation. However, these methods are typically limited to C/C++ and do not explicitly simulate the real-world development or remediation context. In contrast, our work is the first to LLM to synthesize realistic, context-aware function-level vulnerability/fix pairs spanning diverse CWE types and programming languages. By simulating a development and security review workflow, our RVG approach enables high-quality, targeted augmentation of vulnerability datasets.

9 Conclusion and Future Work

In this paper, we address an important gap in automated vulnerability detection by introducing several novel resources and empirical insights. We present BenchVul, a comprehensive, human-verified benchmark for the MITRE 2024 Top 25 Most Dangerous CWEs [17], enabling reliable evaluation of model generalization. We also construct TitanVul, a large-scale, high-quality dataset, disjoint from BenchVul, curated via a multi-agent LLM framework, and propose the RVG framework for synthesizing realistic, context-aware vulnerability data to tackle data scarcity. Our empirical study exposes fundamental weaknesses in existing datasets, such as duplication and CWE imbalance. Experiments show that the performance of training and testing on the same datasets is misleading: only models trained on our high-quality TitanVul achieve robust generalization on BenchVul, establishing a new foundation for trustworthy ML-based vulnerability detection. In future work, we plan to extend our resources to cover a broader range of CWEs, support inter-procedural vulnerability analysis, and assess the applicability of our benchmarks and datasets to large-scale, real-world industrial codebases.

References

[1] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. 30–39.

[2] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2021. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering* 48, 9 (2021), 3280–3296.

[3] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David Wagner. 2023. Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 654–668.

[4] Roland Croft, M Ali Babar, and M Mehdi Kholoosi. 2023. Data quality for software vulnerability datasets. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 121–133.

[5] Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. 2024. Vulnerability detection with code language models: How far are we? *arXiv preprint arXiv:2403.18624* (2024).

[6] Brendan Dolan-Gavitt, Patrick Hulin, Engin Kirda, Tim Leek, Andrea Mambretti, Wil Robertson, Frederick Ulrich, and Ryan Whelan. 2016. Lava: Large-scale automated vulnerability addition. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 110–121.

[7] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. A C/C++ code vulnerability dataset with code changes and CVE summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 508–512.

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

[9] Zeyu Gao, Hao Wang, Yuchen Zhou, Wenyu Zhu, and Chao Zhang. 2023. How far have we gone in vulnerability detection using large language models. *arXiv preprint arXiv:2311.12420* (2023).

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[11] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850* (2022).

[12] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[13] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).

[14] Jingxuan He, Mark Vero, Gabriela Krasnopolska, and Martin Vechev. 2024. Instruction tuning for secure code generation. *arXiv preprint arXiv:2402.09497* (2024).

[15] Yikun Li, Ting Zhang, Ratnadira Widyasari, Yan Naing Tun, Huu Hung Nguyen, Tan Bui, Ivana Clairine Irsan, Yiran Cheng, Xiang Lan, Han Wei Ang, et al. 2024. CleanVul: Automatic Function-Level Vulnerability Detection in Code Commits Using LLM Heuristics. *arXiv preprint arXiv:2411.17274* (2024).

[16] Yu Liu, Lang Gao, Mingxin Yang, Yu Xie, Ping Chen, Xiaojin Zhang, and Wei Chen. 2024. Vuldetectbench: Evaluating the deep capability of vulnerability detection with large language models. *arXiv preprint arXiv:2406.07595* (2024).

[17] MITRE Corporation. 2024. *2024 CWE Top 25 Most Dangerous Software Weaknesses*. https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html

[18] Yu Nong, Yuzhe Ou, Michael Pradel, Feng Chen, and Haipeng Cai. 2022. Generating realistic vulnerabilities via neural code editing: an empirical study. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1097–1109.

[19] Yu Nong, Yuzhe Ou, Michael Pradel, Feng Chen, and Haipeng Cai. 2023. Vulgen: Realistic vulnerability generation via pattern mining and deep learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2527–2539.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[21] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. 155–165.

[22] Niklas Risse and Marcel Böhme. 2024. Uncovering the limits of machine learning for automatic vulnerability detection. In *33rd USENIX Security Symposium (USENIX Security 24)*. 4247–4264.

[23] Niklas Risse, Jing Liu, and Marcel Böhme. 2025. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 388–410.