VulnBench: A Comprehensive Benchmark for Transformer-Based Vulnerability Detection

Jake Norton
University of Otago
norja159@student.otago.ac.nz

David Eyers
University of Otago
david.eyers@otago.ac.nz

Veronica Liesaputra
University of Otago
veronica.liesaputra@otago.ac.nz

Abstract

Reproducible benchmarking for automatically detecting vulnerabilities in source code remains challenging due to inconsistent implementations, varying data preprocessing, and methodological flaws that compromise fair model comparison. In a recent study, 9 in 10 vulnerability detection studies were found to use inappropriate evaluation approaches, with models achieving high scores through spurious correlations rather than actual vulnerability detection. We present VulnBench, an extensible, open-source benchmarking tool that enables fair comparison across models and datasets. Our systematic evaluation of CodeBERT, GraphCodeBERT, CodeT5 (encoder-only and full), and NatGen across nine mostly C/C++ source code datasets reveals that proper threshold optimization can improve F1-scores by up to 54%, as well as large variation in F1-scores showing the large gap in the difficulty of the vulnerability dataset field. By standardising evaluation protocols, VulnBench enables researchers to distinguish between genuine model improvements and methodological artifacts as well as reducing wasteful duplication of effort spent on reproducing results.

Introduction

Software vulnerabilities represent critical security risks that can lead to data breaches, system compromises, and substantial economic losses. Traditional vulnerability detection methods rely heavily on static analysis tools and manual code reviews, which are resource-intensive and often miss complex vulnerability patterns. The emergence of machine learning approaches, particularly deep learning models trained on large code corpora, has shown promise in automating vulnerability detection with improved accuracy and coverage due to these models having much better semantical understanding (Shiri Harzevili et al., 2024).

Recent advances in transformer-based models have revolutionized code understanding tasks. Models like CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021) leverage pre-training on massive code repositories to capture semantic relationships in source code. Graph-enhanced approaches such as GraphCodeBERT (Guo et al., 2021) incorporate structural code information to improve understanding of data flow and control dependencies.

However, evaluating and comparing these models faces significant methodological challenges. As stated Risse, Liu, and Böhme (2025) found that vulnerability detection papers they studied often used inappropriate problem formulations, with models achieving high scores through spurious correlations rather than actual vulnerability detection (Risse, Liu, and Böhme, 2025). Implementation inconsistencies are common, with studies often relying on incomplete model reproductions or subtly different data preprocessing approaches (Varkey, Jiang, and Huang, 2024).

Furthermore, critical methodological considerations are often overlooked. Default classification thresholds (0.5) are frequently suboptimal for the severely imbalanced datasets common in vulnerability detection, yet systematic threshold optimization remains rare (Esposito et al., 2021; Leevy et al., 2023). Recent work specifically examining vulnerability detection datasets confirms that proper threshold selection and loss function choice significantly impact performance on imbalanced data (Ma et al., 2025).

For the reasons stated above, VulnBench aims to provide researchers with a strong starting point to start their exploration into the space.

VulnBench Architecture and Features

VulnBench provides a comprehensive benchmarking platform addressing systematic evaluation challenges in vulnerability detection research. The framework implements four key architectural components:

- **Unified Model Interface:** Provides consistent APIs over diverse transformer architectures. Researchers can easily add new models by implementing a standard interface, with built-in handling of architectural differences (encoder-only vs. encoder-decoder, different tokenization schemes and loss functions).
- **Automated Data Pipeline:** Handles nine vulnerability datasets preprocessed into a standard format. Features will include reproducible seed-based splitting, automatic dataset downloading, consistent function anonymization, and configurable train/validation/test ratios. Function anonymization is important for datasets like Juliet C/C++ (NIST, 2022) as they include information about the code in the function declaration which tells the model how to classify it, e.g. 'void CWE114_Process_Control_bad()'.

Custom datasets will require conversion to a given JSONL format, however, there are many script recipes provided to use as templates.

- **Evaluation Engine:** Implements threshold optimization, multi-seed statistical testing, and comprehensive metrics. Automatically generates performance reports with confidence intervals, statistical significance tests, and comparative analysis.
- **Experiment Tracking:** Integrated Weights & Biases support with standalone logging fallback. Provides real-time training monitoring, hyperparameter comparison, and automated results aggregation across multiple seeds.

Related Work

Code Understanding Benchmarks. CodeXGLUE (Lu et al., 2021) represents the most comprehensive benchmarking effort for code intelligence, providing 14 datasets across 10 tasks including defect detection, with standardized evaluation protocols and baseline models. However, CodeXGLUE focuses broadly on code understanding rather than specifically addressing the methodological challenges in vulnerability detection evaluation.

Vulnerability-Specific Benchmarks. Many frameworks target vulnerability detection specifically. CASTLE (Dubniczky et al., 2025) introduces a curated collection of 250 C programs for evaluating static analyzers, formal verification tools, and LLMs, with balanced vulnerable/non-vulnerable distributions. Often the focus is on a new dataset, not on the tooling around datasets and model training.

Critical Evaluation Issues. Recent work has identified fundamental problems with current evaluation approaches (Risse, Liu, and Böhme, 2025). Studies of popular datasets (BigVul, CVEFixes, DiverseVul) reveal significant data leakage issues and labeling problems that compromise evaluation validity (Ullah et al., 2024).

Benchmarking of Static Analysis Tools. Web service vulnerability detection studies (Antunes and Vieira, 2010) propose benchmarking approaches for comparing penetration testing tools, static analyzers, and anomaly detectors using metrics like precision, recall, and F-measure. Comprehensive evaluations of memory error detectors (Nong et al., 2021) against benchmark datasets reveal varied accuracy across vulnerability categories and highlight the challenges of making fair comparisons.

Dataset Quality and Standardization. Systematic surveys (Shiri Harzevili et al., 2024) reveal that 39.1% of vulnerability detection studies use hybrid data sources combining benchmarks, repositories, and projects, while 37.6% rely on standardized benchmark datasets. However, many proposed benchmarking frameworks (Lin et al., 2019) focus on specific neural network architectures rather than providing systematic evaluation methodology.

Distinguishing Characteristics of VulnBench. Unlike existing work, VulnBench addresses the methodological issues by providing: (1) standardized threshold optimization addressing severe class imbalances, (2) architecture-aware evaluation protocols handling transformer model differences, and (3) systematic dataset quality assessment revealing artifacts that compromise research validity. Existing frameworks serve different purposes: CodeXGLUE provides broad benchmarks for general code understanding tasks, while OWASP Benchmark and CASTLE focus on evaluating specific security tools in controlled environments.

Demonstration Results

Table 1: Best Model Performance per Dataset (F1-Score, 3 seeds)

Dataset	Best Model	F1-Score
CVEFixes	GraphCodeBERT	0.603±0.006
Devign	CodeT5	0.671±0.010
DiverseVul	NatGen	0.307±0.025
Draper	CodeT5	0.609±0.007
ICVul	NatGen	0.586±0.003
Juliet	NatGen	0.900±0.003
Reveal	GraphCodeBERT	0.486±0.007
VulDeepecker	CodeT5	0.959±0.001

VulnBench evaluation across four transformer architectures and eight datasets reveals critical methodological insights:

1. **Threshold optimization universally improves performance**—100% of model-dataset combinations show positive F1 gains (median: +0.082, best: +0.542)
2. **Dataset quality varies dramatically**—synthetic datasets (Juliet, VulDeepecker) achieve F1 > 0.9 while real-world datasets struggle (DiverseVul: 0.307)
3. **Architecture matters**—CodeT5 and NatGen consistently outperform encoder-only models. Complete results and reproducible code available at: https://github.com/ijakenorton/defect_detection_benchmark.

Practical Impact: VulnBench enables researchers to distinguish between genuine model improvements and methodological artifacts. Our evaluation confirms recent findings about dataset quality issues (Ding et al., 2024; Chen et al., 2023). Synthetic datasets in our evaluation (Juliet: F1=0.900, VulDeepecker: F1=0.959) show artificially inflated performance compared to real-world datasets (DiverseVul: F1=0.307, CVEFixes: F1=0.603), consistent with reports of significant data quality problems including poor label accuracy and high duplication rates Ding et al. (2024). While VulnBench does not resolve underlying dataset quality problems—which persist across widely-used benchmarks—it provides transparent evaluation protocols that expose these artifacts rather than obscure them.

Availability and Future Extensions

VulnBench is available as an open-source framework. Future development includes support for additional transformer architectures, or other state-of-the-art models like LineVul or Vul_Detection_Gnn and integration of more datasets e.g. PrimeVul(Ding et al., 2024).

References

Antunes, N., and Vieira, M. 2010. Benchmarking vulnerability detection tools for web services.

Chen, Y.; Ding, Z.; Alowain, L.; Chen, X.; and Wagner, D. 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*, RAID '23, 654–668. New York, NY, USA: Association for Computing Machinery.

Ding, Y.; Fu, Y.; Ibrahim, O.; Sitawarin, C.; Chen, X.; Alomair, B.; Wagner, D.; Ray, B.; and Chen, Y. 2024. Vulnerability detection with code language models: How far are we?

Dubniczky, R. A.; Horvát, K. Z.; Bisztray, T.; Ferrag, M. A.; Cordeiro, L. C.; and Tihanyi, N. 2025. CASTLE: Benchmarking dataset for static code analyzers and LLMs towards cwe detection.

Esposito, C.; Landrum, G. A.; Schneider, N.; Stiefl, N.; and Riniker, S. 2021. Ghost: Adjusting the decision threshold to handle imbalanced data in machine learning. *Journal of Chemical Information and Modeling* 61(6):2623–2640. PMID: 34100609.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547. Association for Computational Linguistics.

Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Liu, S.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; Tufano, M.; Deng, S. K.; Clement, C.; Drain, D.; Sundaresan, N.; Yin, J.; Jiang, D.; and Zhou, M. 2021. Graphcodebert: Pretraining code representations with data flow. In *International Conference on Learning Representations*.

Leevy, J.; Johnson, J.; Hancock, J.; and Khoshgoftaar, T. 2023. Threshold optimization and random undersampling for imbalanced credit card data. *Journal of Big Data* 10.

Lin, G.; Xiao, W.; Zhang, J.; and Xiang, Y. 2019. Deep learning-based vulnerable function detection: A benchmark. In *Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers*, 219–232. Berlin, Heidelberg: Springer-Verlag.

Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C. B.; Drain, D.; Jiang, D.; Tang, D.; Li, G.; Zhou, L.; Shou, L.; Zhou, L.; Tufano, M.; Gong, M.; Zhou, M.; Duan, N.; Sundaresan, N.; Deng, S. K.; Fu, S.; and Liu, S. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. *CoRR* abs/2102.04664.

Ma, X.; He, Y.; Keung, J.; Tan, C.; Ma, C.; Hu, W.; and Li, F. 2025. On the value of imbalance loss functions in enhancing deep learning-based vulnerability detection. *Expert Systems with Applications* 291:128504.

NIST. 2022. Juliet test suite for C/C++ version 1.3.1 with extra support. https://samate.nist.gov/SARD/test-suites/116. Software Assurance Reference Dataset (SARD).

Nong, Y.; Cai, H.; Ye, P.; Li, L.; and Chen, F. 2021. Evaluating and comparing memory error vulnerability detectors. *Information and Software Technology* 137:106614.

Risse, N.; Liu, J.; and Böhme, M. 2025. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection.

Shiri Harzevili, N.; Boaye Belle, A.; Wang, J.; Wang, S.; Jiang, Z. M. J.; and Nagappan, N. 2024. A systematic literature review on automated software vulnerability detection using machine learning. *ACM Comput. Surv.* 57(3).

Ullah, S.; Han, M.; Pujar, S.; Pearce, H.; Coskun, A.; and Stringhini, G. 2024. LLMs cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks.

Varkey, A.; Jiang, S.; and Huang, W. 2024. Codecse: A simple multilingual model for code and comment sentence embeddings.

Wang, Y.; Wang, W.; Joty, S.; and Hoi, S. C. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 8696–8708. Association for Computational Linguistics.