VulnBench: A Comprehensive Benchmark for Transformer-Based Vulnerability Detection

Jake Norton, David Eyers, Veronica Liesaputra
University of Otago
jake.norton@otago.ac.nz, david.eyers@otago.ac.nz, veronica.liesaputra@otago.ac.nz

Abstract

Reproducible benchmarking of tools that automatically detect 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 eight mostly C/C++ source code datasets reveals that proper threshold optimization can improve F1-scores by up to 54%, as well as wide 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 security risks that can lead to critical 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 impractical for detecting 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 the semantic patterns that these models encode (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 (Du and Yu 2023) incorporate structural code information to improve understanding of data flow and control dependencies.

However, evaluating and comparing these models is challenging. (Risse, Liu, and Böhme 2025) found that vulnerability detection papers they studied often used flawed problem formulations, with models achieving high scores through misleading correlations rather than truly detecting vulnerabilities.

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).

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 with built-in handling of architectural differences (encoder-only versus encoder-decoder, different tokenization schemes and loss functions).
- **Automated Data Pipeline:** Handles eight vulnerability datasets preprocessed into a standard format with reproducible seed-based splitting, automatic dataset downloading, and optional function anonymization. While function names and comments can legitimately inform vulnerability prediction in practice, synthetic datasets (e.g., Juliet (NIST 2022)) explicitly encode labels in identifiers (e.g., `CWE114_bad()`), requiring anonymization to prevent trivial classification. Custom datasets require conversion to JSONL format, with script templates provided.
- **Evaluation Engine:** Implements threshold optimization, multi-seed statistical testing, and comprehensive metrics. Automatically generates performance reports with confi-

dence 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.

Threshold Optimization Methodology

VulnBench supports two threshold optimization strategies: basic grid search and GHOST (Generalized tHreshOld ShifTing) (Esposito et al. 2021). The grid search approach evaluates candidate thresholds (0.1 to 0.9, step 0.02) once on the validation set, selecting the threshold maximizing F1-score. Results presented in this paper use grid search. GHOST provides a more robust alternative through bootstrap aggregation: it evaluates the same 40 thresholds on 100 random subsets of the validation set (80% sampling without replacement), selecting the threshold with highest median Cohen's Kappa across subsets. This reduces overfitting by accounting for data variability. Both methods require minimal computational overhead (approximately 2-3 minutes per model) and re-optimize thresholds per model-dataset combination rather than using fixed values.

Related Work

Code Understanding and Vulnerability Benchmarks. CodeXGLUE (Lu et al. 2021) provides comprehensive benchmarks for code intelligence across 14 datasets and 10 tasks including defect detection, though it focuses broadly on code understanding rather than vulnerability-specific evaluation challenges. CASTLE (Dubniczky et al. 2025) targets vulnerability detection specifically with 250 curated C programs for evaluating static analyzers and LLMs.

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). Systematic surveys (Shiri Harzevili et al. 2024) show that while 37.6% of studies rely on standardized benchmarks, many proposed frameworks (Lin et al. 2019) focus on specific architectures rather than systematic evaluation methodology.

Distinguishing Characteristics of VulnBench. Unlike existing work, VulnBench addresses 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

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

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

Table 1: Best F1-Score Model per Dataset, using a random 80/10/10 train–validation–test split with three unique seeds.

(complete results and reproducible code online[1]):

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.

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. 2025; Chen et al. 2023). Synthetic datasets (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 (Ding et al. 2025). VulnBench provides transparent evaluation protocols that expose these artifacts.

Availability and Future Extensions

VulnBench is available as an open-source framework. Future development will include: support for additional transformer architectures; other state-of-the-art models like LineVul (Fu and Tantithamthavorn 2022); integration of more datasets e.g., PrimeVul (Ding et al. 2025).

Acknowledgements

This research was supported by the Ministry of Business, Innovation and Employment (MBIE), New Zealand. The author gratefully acknowledges this funding, which made this work possible. We also acknowledge the open-source community for providing the datasets and tools that enabled this evaluation framework.

References

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*

[1]https://github.com/ijakenorton/VulnBench

in Attacks, Intrusions and Defenses, RAID '23, 654–668. New York, NY, USA: Association for Computing Machinery. ISBN 9798400707650.

Ding, Y.; Fu, Y.; Ibrahim, O.; Sitawarin, C.; Chen, X.; Alomair, B.; Wagner, D.; Ray, B.; and Chen, Y. 2025. Vulnerability Detection with Code Language Models: How Far are We? In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 1729–1741.

Du, Y.; and Yu, Z. 2023. Pre-training Code Representation with Semantic Flow Graph for Effective Bug Localization. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, 579–591. New York, NY, USA: Association for Computing Machinery. ISBN 9798400703270.

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. arXiv:2503.09433.

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.

Fu, M.; and Tantithamthavorn, C. 2022. LineVul: A Transformer-based Line-Level Vulnerability Prediction. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 608–620.

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. ISBN 978-3-030-41578-5.

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).

Risse, N.; Liu, J.; and Böhme, M. 2025. Top Score on the Wrong Exam: On Benchmarking in Machine Learning for Vulnerability Detection. *Proc. ACM Softw. Eng.*, 2(ISSTA).

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 . In *2024 IEEE Symposium on Security and Privacy (SP)*, 862–880. Los Alamitos, CA, USA: IEEE Computer Society.

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.