Transformer-Based Approaches for Source Code Vulnerability Detection

Koushik Ahamed Tonmoy [1] **Afridi Hassan** [1] **Shahriyar Zaman Ridoy** [1]

Abstract

Detecting software vulnerabilities is critical for the security of systems written in memory-unsafe languages such as C and C++, yet traditional static analyzers rely on rigid rules and syntactic pattern matching that generalize poorly across codebases and produce many false positives. We study pretrained code transformers as a data-driven alternative for multi-class vulnerability classification on the Draper VDISC dataset, comparing three pipelines: classical classifiers trained over frozen code embeddings, end-to-end fine-tuning of the transformers, and ensemble stacking of model logits through a meta-classifier. We evaluate CodeBERT, GraphCodeBERT, and UniXcoder, which differ in their pretraining objectives and inductive biases. Fine-tuning substantially outperforms frozen and classical baselines, with UniXcoder achieving the strongest single-model results. Logit-stacking ensembles, particularly UniXcoder paired with GraphCodeBERT, improve further to 82.9% accuracy. Our findings highlight the value of architectural diversity for static vulnerability detection and provide a reproducible benchmark across four modeling strategies.

1. Introduction

Software security remains a central concern in modern computing, with the number of disclosed vulnerabilities growing steadily across both open-source and proprietary codebases. Languages such as C and C++ offer low-level control and high performance but are particularly susceptible to memory-related flaws, including buffer overflows, invalid pointer dereferences, and integer overflows (Szekeres et al., 2013). The consequences range from crashes to full system compromise. Static analysis tools are widely deployed to surface such issues, but they typically depend on hand-written heuristics and syntactic pattern matching. As a result, they are brittle, prone to high false-positive rates, and struggle to generalize to novel or obfuscated code (Aloraini et al., 2019).

The rise of transformer models pretrained on source code offers a data-driven alternative. Earlier neural and representation-learning approaches showed that learned models can capture vulnerability-relevant program patterns beyond hand-written rules, including code gadgets, program graphs, and transformer-based code representations (Li et al., 2018; Yamaguchi et al., 2014; Zhou et al., 2019; Fu & Tantithamthavorn, 2022; Hanif & Maffeis, 2022). Models such as CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), and UniXcoder (Guo et al., 2022) extend masked-language pretraining to programming languages, learning representations that capture not only lexical tokens but also semantic context, data and control flow, and structural relationships. These representations are promising for downstream tasks such as defect prediction, code summarization, and vulnerability classification.

In this work, we systematically evaluate code transformers for source code vulnerability detection on the Draper VDISC dataset (Russell et al., 2018), a large labeled corpus of C/C++ functions annotated by Common Weakness Enumeration (CWE) type. Our framework comprises four progressively more expressive strategies: (1) classical classifiers over frozen transformer embeddings; (2) lightweight neural heads over frozen features; (3) end-to-end fine-tuning of the full transformer; and (4) ensemble learning that stacks model logits through a meta-classifier. Beyond benchmarking accuracy and weighted F1-score, we examine the trade-offs in training cost and the ability to detect rare vulnerability classes, offering practical guidance for integrating learned models into security-auditing pipelines.

2. Methodology

We design a multi-stage pipeline that progressively increases model expressiveness, applying consistent preprocessing, downsampling, and evaluation protocols across all stages, using the Draper VDISC dataset to ensure a fair comparison.

Classical learning on static embeddings. We ex-

[1]Department of Electrical and Computer Engineering, North South University, Dhaka, Bangladesh. Correspondence to: Koushik Ahamed Tonmoy <koushik.tonmoy@northsouth.edu>, Shahriyar Zaman Ridoy <shahriyar.zaman01@gmail.com>.

Published at the 6th Muslims in ML (MusIML) Workshop, co-located with ICML 2026, Seoul, South Korea, July, 2026.

tract 768-dimensional representations of C/C++ functions from the pretrained CodeBERT model via its `[CLS]` token and treat them as fixed features. Three gradient-boosting classifiers, XGBoost (Chen & Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), and LightGBM (Ke et al., 2017), and a simple multilayer perceptron (MLP) are trained on these embeddings. This stage establishes a low-cost baseline for measuring how far static representations can go without fine-tuning.

Lightweight neural classification over frozen transformers. We freeze each of CodeBERT, GraphCodeBERT, and UniXcoder and train a two-layer MLP head on the extracted `[CLS]` features. The three encoders differ in their pretraining: CodeBERT is trained bidirectionally on code and natural language; GraphCodeBERT additionally incorporates data-flow structure; and UniXcoder supports cross-modal learning with code, comments, and abstract syntax trees. This stage isolates the representational capacity of each frozen encoder.

End-to-end fine-tuning. We fine-tune the full transformer by initializing its weights with pretrained weights from the Hugging Face Transformers library (Wolf et al., 2020), attaching a classification head, and training with cross-entropy loss. CodeBERT and GraphCodeBERT use a custom training loop for fine-grained control over the optimizer and schedule; UniXcoder uses the standardized Hugging Face `Trainer` API. Fine-tuning lets each model specialize in dataset-specific patterns.

Ensembling via stacked generalization. We fine-tune each model individually and extract pre-softmax logits on the validation and test splits. Following stacked generalization (Wolpert, 1992), the logits of two models (i) UniXcoder + CodeBERT and (ii) UniXcoder + GraphCodeBERT are concatenated and fed to a logistic-regression meta-classifier trained on validation logits and evaluated on test logits. The design exploits complementary inductive biases: GraphCodeBERT captures structural dependencies while UniXcoder captures semantic relationships.

Across all stages, we use model-specific Hugging Face tokenizers with a maximum sequence length of 512, a training batch size of 16 (inference 32), and the AdamW optimizer (Loshchilov & Hutter, 2019) with early stopping on validation loss. We report accuracy and weighted F1-score as primary metrics, supplemented by class-wise precision and recall. Full hyperparameters are listed in Section A.

3. Dataset and Preprocessing

We use the Draper VDISC dataset (Russell et al., 2018), which contains over 1.27 million C/C++ functions drawn from public repositories, each labeled by static analysis into one of five CWE categories: CWE-119 (memory), CWE-120 (buffer overflow), CWE-469 (pointer subtraction), CWE-476 (NULL pointer dereference), and a residual CWE-other class.

Table 1. Class distribution after null removal and downsampling. CWE-469 remains severely underrepresented.

Class	Train	Val.	Test
CWE-119 (memory)	5,942	1,142	1,142
CWE-120 (buffer ovf.)	5,777	1,099	1,099
CWE-469 (ptr. sub.)	249	53	53
CWE-476 (NULL ptr.)	2,755	535	535
CWE-other	5,582	1,071	1,071
Total	**20,305**	**3,900**	**3,900**

The raw label distribution is heavily skewed toward CWE-119 and CWE-120. We remove null entries and apply random downsampling to reduce the dominance of the majority classes while retaining sufficient training data, following standard concerns in learning from imbalanced data (He & Garcia, 2009). Each class is integer-encoded (0–4), and code is tokenized with padding and truncation to 512 tokens. The resulting data follow an $80/10/10$ train/validation/test split. As shown in Table 1, CWE-469 remains an extreme minority, accounting for only 249 training examples and 53 examples in each evaluation split.

4. Experiments and Results

We evaluate classical baselines, frozen transformer encoders, fine-tuned transformers, and stacked-logit ensembles under identical train/validation/test splits and a common evaluation protocol (Section A). We report accuracy and weighted F1 as headline metrics and analyze class-wise F1 to expose behavior on the imbalanced label distribution. The complete per-model tables that support every claim in this section are provided in Section B.

Overall Comparison Across Strategies. We first compare the four modeling strategies at the aggregate level, using the best or representative model from each. Figure 2 reports test accuracy and weighted F1 side by side and reveals a clear, monotonic progression in capability. Classical learners over frozen CodeBERT embeddings form the weakest tier: LightGBM, the strongest among them, reaches only 0.587 accuracy and 0.583 weighted F1, indicating that static `[CLS]` features are not directly separable enough for reliable vulnerability classification. Replacing the classical head with a learned MLP over a frozen transformer yields only a marginal gain—GraphCodeBERT improves to 0.604 accuracy— which shows that a stronger pretrained encoder, used as a fixed feature extractor, does not by itself resolve the task.

The decisive jump comes from end-to-end fine-tuning,

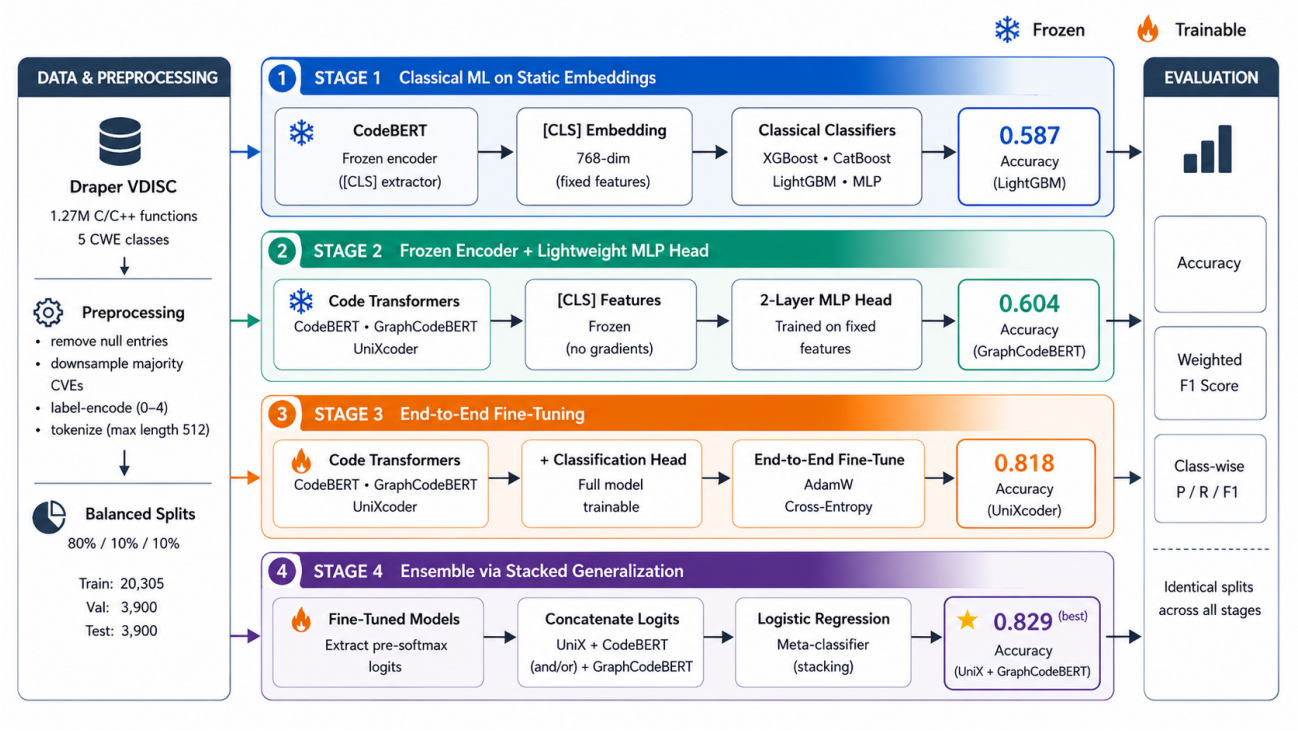

Figure 1. Methodology overview. The pipeline evaluates four progressively stronger strategies for source-code vulnerability detection on Draper VDISC: classical classifiers over frozen CodeBERT embeddings, frozen code transformers with an MLP head, end-to-end transformer fine-tuning, and stacked-logit ensembling. All stages use the same preprocessing, tokenization, data splits, and evaluation protocol.

where all transformer parameters adapt to the Draper VDISC label space. Fine-tuned CodeBERT and GraphCodeBERT both rise to roughly 0.766 accuracy, while UniXcoder reaches 0.818 accuracy and 0.816 weighted F1, the best single model owing to its cross-modal pretraining over code, comments, and abstract syntax trees. Stacked-logit ensembling adds a final, consistent improvement: UniXcoder + CodeBERT attains 0.827 accuracy, and the UniXcoder + GraphCodeBERT ensemble (highlighted in Figure 2) achieves the strongest overall result of 0.829 accuracy and 0.828 weighted F1. The near-identical accuracy and weighted-F1 values within each model confirm that the headline gains are not an artifact of class support, and the ≈ 24-point spread from classical baselines to the best ensemble underscores that task-specific adaptation, rather than representation reuse, drives performance on this benchmark.

Per-Class Behavior Across Learning Stages. The aggregate numbers hide a sharper story at the class level. To make the source of the gains explicit, Table 2 traces test F1 for every CWE class through the three transformer-based stages, using the best model at each stage. Three distinct trajectories emerge. The two majority classes, CWE-119 and CWE-120, are already moderately learned by the frozen encoder (0.679 and 0.625 F1) and improve smoothly with adaptation, ending at 0.883 and 0.872 for the best ensemble. The mid-frequency classes CWE-476 and CWE-other benefit the most from fine-tuning in absolute terms, jumping from 0.560 and 0.540 under the frozen encoder to 0.779 and 0.755 once the encoder is adapted—an increase of roughly 0.22 F1 that the frozen representation never reaches. This indicates that the cues separating NULL pointer dereferences and the residual class from memory and buffer errors are present in the data but are recovered only when the encoder is allowed to specialize.

The rare CWE-469 class follows a third trajectory entirely and is the persistent failure mode. Classical models, all frozen encoders, and even the fine-tuned CodeBERT and GraphCodeBERT obtain an F1 of exactly 0.00 on this class (Section B): they never predict it. Only fine-tuned UniXcoder recovers it at all (0.386 F1), and the ensembles lift it slightly further to 0.400. Strong macro-level performance on imbalanced vulnerability data therefore hinges on a model's ability to represent scarce CWE types, a known challenge in imbalanced learning (He & Garcia, 2009), and architectural choice (here, UniXcoder's cross-modal pretraining) matters more than ensembling for this hardest case.

Generalization and the Macro–Weighted Gap Two diagnostics confirm that these results reflect genuine learning rather than artifacts of the evaluation setup. First, wherever

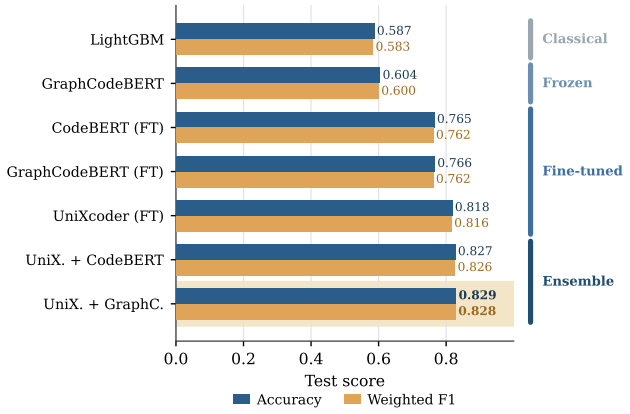

Figure 2. Test accuracy and weighted F1 across the four modeling strategies, using the best or representative model per strategy. Models are ordered by increasing capability and grouped by strategy (right-hand brackets: classical learners, frozen encoders, fine-tuned transformers, and stacked-logit ensembles). Each successive strategy improves over the previous one; the UniXcoder + GraphCodeBERT ensemble (highlighted row) is strongest at 0.829 accuracy / 0.828 weighted F1. Per-model and class-wise numbers are reported in Section B.

Table 2. Class-wise test F1 for the best model at each transformer-based stage: frozen GraphCodeBERT, fine-tuned UniXcoder, and the UniXcoder + GraphCodeBERT ensemble. Fine-tuning drives the largest gains on the mid-frequency classes, while the rare CWE-469 class is recovered only after fine-tuning.

Class	Frozen	Fine-tuned	Ensemble
CWE-119 (memory)	0.679	0.875	0.883
CWE-120 (buffer ovf.)	0.625	0.855	0.872
CWE-469 (ptr. sub.)	0.000	0.386	0.400
CWE-476 (NULL ptr.)	0.560	0.779	0.788
CWE-other	0.540	0.755	0.765
Macro F1	0.481	0.730	0.741
Weighted F1	0.600	0.816	0.828

both splits are reported, validation and test scores agree to within about one point—frozen GraphCodeBERT scores 0.6036 validation against 0.6044 test accuracy, and classical LightGBM 0.5805 against 0.5867—so the early-stopping protocol in Section A yields models that generalize rather than overfit the validation set. Second, and more diagnostic, is the gap between weighted and macro F1. For the classical and frozen tiers this gap is large (≈ 0.12; weighted 0.58–0.60 against macro 0.46–0.48), because metrics that ignore class support are dragged down by the 0.00 F1 on CWE-469. Fine-tuned UniXcoder narrows the gap to 0.086 (macro 0.730 versus weighted 0.816), and the ensembles hold it near 0.087. The macro–weighted gap thus serves as a compact proxy for how heavily a model leans on the majority classes, and its collapse coincides exactly with the first recovery of the rare class.

Why Frozen UniXcoder Underperforms A counterintuitive result deserves emphasis: frozen UniXcoder is the weakest encoder of all (0.4928 test F1), trailing even classical learners on frozen CodeBERT embeddings (0.5827), despite becoming the strongest model once fine-tuned. We attribute this reversal to a representation–objective mismatch. UniXcoder's unified cross-modal pretraining optimizes a representation suited to generation and retrieval, whose pooled embedding is not directly linearly separable for a lightweight discriminative head. The explicit data-flow signal in GraphCodeBERT, by contrast, transfers more readily to the frozen-feature setting, which is why it leads that tier. Only when the full encoder is adapted does UniXcoder's richer pretraining translate into the best single-model performance. The practical lesson is that frozen-feature quality is a poor predictor of fine-tuned capability for this task, and that probing experiments alone could mislead model selection.

What Ensembling Does and Does Not Fix Finally, the stacked-logit ensembles show that architectural complementarity helps, but only modestly. UniXcoder + CodeBERT reaches 0.8269 test accuracy, while UniXcoder + GraphCodeBERT reaches 0.8285. The gains mainly appear in well-represented classes, such as CWE-120 ($0.855 \rightarrow 0.872$) and CWE-other ($0.755 \rightarrow 0.765$), suggesting that stacking refines decision boundaries where both base models already have sufficient supervision. In contrast, the rare CWE-469 class improves only slightly ($0.386 \rightarrow 0.400$). Thus, ensembling improves calibration and common class performance, but the rare-class gap likely requires imbalance-aware objectives, targeted resampling, or data augmentation rather than further model combination (He & Garcia, 2009).

5. Conclusion

We presented a comparative study of classical machine learning and pretrained code transformers for multi-class classification of source code vulnerabilities. Static embeddings with classical classifiers establish weak baselines; frozen transformer encoders improve modestly; and end-to-end fine-tuning delivers large gains, with UniXcoder the strongest single model owing to its cross-modal pretraining. Logit-stacking ensembles that combine UniXcoder with CodeBERT or GraphCodeBERT outperform all single-model baselines, confirming the benefits of architectural diversity. Across all configurations, the rare CWE-469 class remains the dominant bottleneck and is recovered only by the most capable models. Beyond benchmarking, our staged methodology offers a reproducible template for evaluating learned vulnerability detectors, and points to imbalance-aware training as the most promising direction for future work.

Impact Statement

This work aims to improve the automated detection of security vulnerabilities in source code, helping developers identify and remediate flaws earlier and reduce the incidence of exploitable software. Learned vulnerability detectors are not perfect; false negatives may leave real vulnerabilities undetected, and false positives may waste auditing efforts. Therefore, such models should augment, not replace, human review and established static analysis practices (Aloraini et al., 2019; Chakraborty et al., 2022). Because the underlying labels are produced by static analysis tools, models trained on them inherit those tools' biases and blind spots, and reported performance should not be read as a guarantee of safety (Aloraini et al., 2019). We use only publicly available, non-personal code data and release a reproducible methodology to support scrutiny and extension by the community.

Cost and Reproducibility Statement

The gains are not free. Classical classifiers train in seconds and frozen-encoder heads in roughly two minutes, whereas full fine-tuning and ensembling require GPU training over many epochs. In settings where latency or compute is constrained, a fine-tuned single model (UniXcoder) captures most of the benefit; ensembling buys a final ≈ 1 accuracy point at the cost of training and serving two transformers. Code, experiment configurations, processed-split details, and reproduction instructions for all classical, frozen-transformer, fine-tuned-transformer, and stacked-logit ensemble experiments are available at `https://github.com/koushiktonmoy/codevul`.

References

Aloraini, B., Nagappan, M., German, D. M., Hayashi, S., and Higo, Y. An empirical study of security warnings from static application security testing tools. *Journal of Systems and Software*, 158:110427, 2019. doi: 10.1016/j.jss.2019.110427.

Chakraborty, S., Krishna, R., Ding, Y., and Ray, B. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48(9):3280–3296, 2022.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. doi: 10.1145/2939672.2939785.

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

Fu, M. and Tantithamthavorn, C. LineVul: A transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pp. 608–620, 2022. doi: 10.1145/3524842.3528452.

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. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations (ICLR)*, 2021.

Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., and Yin, J. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7212–7225, 2022.

Hanif, H. and Maffeis, S. VulBERTa: Simplified source code pre-training for vulnerability detection. In *2022 International Joint Conference on Neural Networks*, pp. 1–8. IEEE, 2022. doi: 10.1109/IJCNN55064.2022.9892280.

He, H. and Garcia, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. doi: 10.1109/TKDE.2008.239.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., and Zhong, Y. VulDeePecker: A deep learning-based system for vulnerability detection. In *Proceedings of the Network and Distributed System Security Symposium*, 2018.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. CatBoost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31, pp. 6639–6649, 2018.

Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., and McConley, M. Automated vulnerability detection in source code using deep representation learning. In *17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 757–762, 2018.

Szekeres, L., Payer, M., Wei, T., and Song, D. SoK: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, pp. 48–62. IEEE, 2013. doi: 10.1109/SP.2013.13.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-demos.6.

Wolpert, D. H. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. doi: 10.1016/S0893-6080(05)80023-1.

Yamaguchi, F., Golde, N., Arp, D., and Rieck, K. Modeling and discovering vulnerabilities with code property graphs. In *2014 IEEE Symposium on Security and Privacy*, pp. 590–604. IEEE, 2014. doi: 10.1109/SP.2014.44.

Zhou, Y., Liu, S., Siow, J., Du, X., and Liu, Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pp. 10197–10207, 2019.

A. Training Configuration

This appendix provides the implementation and evaluation details supporting the results reported in the main paper. The goal is to make the comparison across classical learners, frozen transformer encoders, fine-tuned transformer models, and stacked-logit ensembles reproducible under a common experimental protocol. All stages use the same processed Draper VDISC splits, the same label encoding, and the same evaluation metrics so that differences in performance can be attributed primarily to the modeling strategy rather than to changes in data preparation or evaluation setup.

All experiments were run with GPU acceleration where transformer inference or fine-tuning was required. We instantiate the corresponding Hugging Face tokenizer for each model using `AutoTokenizer` and pad or truncate input sequences to a maximum length of 512 tokens. Training uses a batch size of 16, and inference uses a batch size of 32. For end-to-end transformer fine-tuning, we optimize the classification objective using AdamW with early stopping on validation loss. Stage 3 fine-tunes the full encoder with an attached linear classification head under cross-entropy loss; CodeBERT and GraphCodeBERT use a custom training loop, while UniXcoder uses the Hugging Face `Trainer` API. Stage 4 trains a logistic-regression meta-classifier on validation-split logits and evaluates the trained meta-classifier on held-out test logits.

The staged configuration is intentionally conservative: frozen-feature methods test whether pretrained representations are directly separable for vulnerability classification, while fine-tuned models test whether task-specific adaptation is necessary. The stacked-logit ensembles then test whether models with different pretraining biases provide complementary evidence at prediction time.

B. Full Per-Model Result Tables

This section reports the complete per-model results that support the summarized findings in the main text. We include both aggregate metrics and class-wise results because source-code vulnerability detection is not only a standard multi-class classification problem but also an imbalanced security classification problem. In this setting, a model's practical utility depends not only on overall accuracy but also on whether it can recover rare vulnerability classes instead of collapsing to majority-class predictions.

Across the tables, class indices 0–4 correspond to CWE-119, CWE-120, CWE-469, CWE-476, and CWE-other, respectively. Accuracy measures the fraction of correct predictions, while F1-score balances precision and recall. Macro averages treat all classes equally and are therefore sensitive to rare-class failure. Weighted averages account for class support and are therefore closer to the headline dataset-level performance.

B.1. Classical Machine Learning on Static CodeBERT Embeddings

The first group of experiments evaluates whether frozen CodeBERT representations are already sufficiently discriminative for vulnerability classification when paired with strong non-neural classifiers. We extract the `[CLS]` embedding for each function and train XGBoost, CatBoost, LightGBM, and an MLP classifier on these fixed vectors. This setting is computationally inexpensive because the transformer is used only as a feature extractor and no encoder parameters are updated.

Table 3 shows that classical learners over static CodeBERT embeddings provide useful but limited baselines. LightGBM achieves the strongest performance among the classical learners, reaching 0.5867 test accuracy and 0.5827 test F1. However, all three gradient-boosting methods fail completely on Class 2, corresponding to the rare CWE-469 category. This indicates that frozen CodeBERT embeddings alone do not make the minority class linearly or tree-separably recoverable under the current training distribution.

The gap between weighted and macro averages is especially informative. Weighted F1 remains around 0.58 for the best classical model because performance on larger classes dominates the metric. Macro F1 is lower because the rare class receives equal weight and is not recovered by any classical learner. These results establish a low-cost but weak baseline and motivate the transition to transformer-specific classifiers.

B.2. Frozen Transformer Encoders (No Fine-tuning)

The second group of experiments tests whether stronger pretrained encoders improve performance when the encoder itself remains frozen. Unlike the previous setting, this comparison includes CodeBERT, GraphCodeBERT, and UniXcoder, each

Table 3. Classification performance using classical learners over frozen CodeBERT `[CLS]` embeddings. Class indices 0–4 correspond to CWE-119, CWE-120, CWE-469, CWE-476, and CWE-other.

Metric	XGBoost	CatBoost	LightGBM
Overall			
Training Time (s)	12.80	4.75	29.87
Validation Accuracy	0.5828	0.5436	0.5805
Validation F1	0.5778	0.5386	0.5750
Test Accuracy	0.5744	0.5521	0.5867
Test F1	0.5706	0.5477	0.5827
Validation P / R / F1 by class			
Class 0	0.623 / 0.673 / 0.647	0.598 / 0.651 / 0.623	0.618 / 0.689 / 0.652
Class 1	0.586 / 0.616 / 0.600	0.548 / 0.561 / 0.555	0.588 / 0.597 / 0.592
Class 2	0.000 / 0.000 / 0.000	0.000 / 0.000 / 0.000	0.000 / 0.000 / 0.000
Class 3	0.668 / 0.469 / 0.551	0.558 / 0.467 / 0.509	0.639 / 0.473 / 0.544
Class 4	0.508 / 0.539 / 0.523	0.471 / 0.476 / 0.474	0.509 / 0.530 / 0.520
Macro Avg (P/R/F1)	0.477 / 0.459 / 0.464	0.435 / 0.431 / 0.432	0.471 / 0.458 / 0.462
Weighted Avg (P/R/F1)	0.579 / 0.583 / 0.578	0.536 / 0.544 / 0.539	0.574 / 0.581 / 0.575
Support (total)	3900	3900	3900

paired with a lightweight MLP classification head. This isolates the quality of the pretrained representation from the effect of full task-specific fine-tuning.

Table 4 shows that frozen transformer encoders improve only modestly over classical learning on fixed CodeBERT embeddings. GraphCodeBERT achieves the best frozen-encoder test F1 of 0.6003, followed closely by CodeBERT at 0.5942. Frozen UniXcoder underperforms in this setting, reaching only 0.4928 test F1. This suggests that UniXcoder's representation is not automatically superior when used as a fixed encoder; its advantage emerges later when the full model is fine-tuned.

The frozen-encoder results reveal two important patterns. First, simply replacing CodeBERT with a structurally richer pretrained encoder does not solve the rare-class problem: all frozen encoders still produce 0.00 F1 on Class 2. Second, the gap between frozen and fine-tuned models in the next subsection indicates that vulnerability classification requires task adaptation rather than relying solely on generic code representations. This is especially relevant for static vulnerability detection, where small token-level or structural differences may determine the target class.

B.3. Fine-tuned Transformer Models

The third group of experiments evaluates end-to-end fine-tuning. In contrast to the frozen setting, all transformer parameters are updated using the vulnerability classification objective. This allows each model to adapt its internal representations to the Draper VDISC label space rather than relying only on generic pretraining.

Table 5 shows a large performance jump from frozen encoders to fine-tuned transformers. CodeBERT and GraphCodeBERT both improve to approximately 0.766 test accuracy, while UniXcoder reaches 0.8179 test accuracy and 0.8164 test F1. The improvement is not only visible in weighted metrics: UniXcoder also achieves the strongest macro F1, indicating better class-level balance than the other fine-tuned models.

The most important difference appears on Class 2. Fine-tuned CodeBERT and GraphCodeBERT still fail to predict this rare class, producing 0.00 F1. UniXcoder is the only single model that recovers Class 2, reaching 0.386 F1. This result explains why UniXcoder's macro average is substantially higher than the other fine-tuned models, despite all three models performing well on the majority of classes. The finding supports the main paper's claim that rare-class vulnerability detection is the central bottleneck in this benchmark.

B.4. Ensemble Models (Stacked Logits)

The final group of experiments evaluates stacked-logit ensembles. Instead of averaging predictions directly, we concatenate the validation and test logits from two fine-tuned models and train a logistic regression meta-classifier on the validation

Table 4. Classification performance for frozen CodeBERT, GraphCodeBERT, and UniXcoder encoders with a lightweight MLP head. Class-wise entries are reported as Validation / Test F1.

Metric	CodeBERT	GraphCodeBERT	UniXcoder
Overall			
Training Time (s)	116.24	112.99	112.95
Validation Accuracy	0.6082	0.6036	0.4949
Validation F1	0.5989	0.5980	0.4886
Test Accuracy	0.6028	0.6044	0.4995
Test F1	0.5942	0.6003	0.4928
Macro Avg P / R / F1			
Validation	0.481 / 0.486 / 0.481	0.508 / 0.468 / 0.476	0.412 / 0.383 / 0.387
Test	0.478 / 0.486 / 0.480	0.507 / 0.473 / 0.481	0.415 / 0.387 / 0.392
Weighted Avg P / R / F1			
Validation	0.596 / 0.608 / 0.599	0.611 / 0.604 / 0.598	0.500 / 0.495 / 0.489
Test	0.590 / 0.603 / 0.594	0.611 / 0.604 / 0.600	0.504 / 0.500 / 0.493
Class-wise F1 (Validation / Test)			
Class 0	0.703 / 0.692	0.680 / 0.679	0.559 / 0.558
Class 1	0.617 / 0.610	0.623 / 0.625	0.477 / 0.480
Class 2	0.000 / 0.000	0.000 / 0.000	0.000 / 0.000
Class 3	0.575 / 0.600	0.527 / 0.560	0.414 / 0.430
Class 4	0.511 / 0.501	0.550 / 0.540	0.487 / 0.494

logits. This tests whether different transformer architectures encode complementary decision signals that can be exploited by a simple second-stage classifier.

Table 6 shows that both ensembles improve over the best single model. The UniXcoder + CodeBERT ensemble reaches 0.8269 test accuracy and 0.8259 test F1, while the UniXcoder + GraphCodeBERT ensemble achieves the best overall result with 0.8285 test accuracy and 0.8277 test F1. The improvement is modest but consistent, suggesting that stacked logits provide additional value after strong fine-tuning.

At the class level, the ensembles slightly improve the rare CWE-469 class from UniXcoder's 0.386 F1 to 0.400 F1. The larger gains appear on the majority and mid-frequency classes, especially Class 1 and Class 4. This suggests that ensembling mainly improves calibration and boundary refinement across common classes, while rare-class recovery remains limited by the small number of available CWE-469 examples. Therefore, although stacked-logit ensembling gives the best overall performance, future gains are likely to require imbalance-aware objectives, data augmentation, or targeted sampling strategies for the rare vulnerability class.

Table 5. Test performance of fine-tuned CodeBERT, GraphCodeBERT, and UniXcoder. Fine-tuning substantially improves performance over frozen encoders, with UniXcoder producing the strongest single-model result.

Metric	CodeBERT (FT)	GraphCodeBERT (FT)	UniXcoder (FT)
Test Accuracy	0.7654	0.7659	0.8179
Test F1	0.7615	0.7620	0.8164
Macro Avg (P/R/F1)	0.609 / 0.607 / 0.607	0.613 / 0.608 / 0.609	0.750 / 0.717 / 0.730
Weighted Avg (P/R/F1)	0.760 / 0.765 / 0.762	0.761 / 0.766 / 0.762	0.816 / 0.818 / 0.816
Class-wise F1 (Test)			
Class 0	0.873	0.871	0.875
Class 1	0.812	0.806	0.855
Class 2	0.000	0.000	0.386
Class 3	0.640	0.699	0.779
Class 4	0.667	0.671	0.755

Table 6. Test performance of stacked-logit ensembles with a logistic-regression meta-classifier. The UniXcoder + GraphCodeBERT ensemble achieves the strongest overall result.

Metric	UniXcoder + CodeBERT	UniXcoder + GraphCodeBERT
Test Accuracy	0.8269	0.8285
Test F1	0.8259	0.8277
Macro Avg (P/R/F1)	0.768 / 0.723 / 0.740	0.769 / 0.725 / 0.741
Weighted Avg (P/R/F1)	0.826 / 0.827 / 0.826	0.828 / 0.829 / 0.828
Class-wise F1 (Test)		
Class 0	0.884	0.883
Class 1	0.869	0.872
Class 2	0.400	0.400
Class 3	0.785	0.788
Class 4	0.762	0.765