Data Science : Journal Of Computing And Applied Informatics
Journal homepage: https://jocai.usu.ac.id

Size-Controlled Opcode Ablation for Smart Contract Vulnerability Detection

Astrid Pranadani[1], Dhani Ariatmanto[2]
[1,2] *Faculty of Computer Science, Universitas AMIKOM Yogyakarta, Yogyakarta, 55283, Indonesia*
Corresponding Author: Astrid Pranadani (apranadani@gmail.com)

ARTICLE INFO
Article history:
Received 8 July 2026
Revised 28 July 2026
Accepted 29 July 2026
Available online 31 July 2026

E-ISSN: 2580-829X
P-ISSN: 2580-6769

How to cite:
Astrid Pranadani and Dhani Ariatmanto, "Size-Controlled Opcode Ablation for Smart Contract Vulnerability Detection," Data Science : Journal of Computing and Applied Informatics, vol. V10, no. 2, Jul. 2026, doi: 10.32734/jocai.v10i2.26398

ABSTRACT
Smart contract vulnerability detection requires evaluation protocols that separate real representation signal from dataset-specific artifacts. DIVE dataset provides lifecycle-based tabular features for Ethereum smart contracts; however, benchmark performance alone cannot show whether a dominant feature group is useful or only benefits from having many columns. This study examines Opcode Distribution features using 22,330 contracts, 397 processed features, and eight Decentralized Application Security Project (DASP)-aligned vulnerability labels. Five multi-label learning configurations were evaluated under 3×5 repeated cross-validation, followed by global feature-group ablation, size-controlled random opcode ablation, per-label degradation analysis, cumulative stability analysis, and opcode-profile group-aware robustness checking. MultiOutput LightGBM achieved the best baseline performance, with Micro-F1 of 0.91396, Macro-F1 of 0.82464, and Macro-PR-AUC of 0.90146. Removing the full Opcode Distribution group reduced Macro-F1 to 0.78745, while removing a same-sized random opcode subset produced Macro-F1 of 0.82404. The findings indicate that Opcode Distribution acts as a collective predictive representation rather than a feature-count artifact, without implying causal vulnerability mechanisms.
Keywords: ablation study, DIVE Dataset, multi-label classification, opcode distribution, smart contract vulnerability detection.

ABSTRAK
Deteksi kerentanan smart contract membutuhkan protokol evaluasi yang mampu memisahkan sinyal representasi dari artefak spesifik dataset. Dataset DIVE menyediakan fitur lifecycle untuk kontrak Ethereum, tetapi performa benchmark belum menunjukkan apakah kelompok fitur dominan benar-benar berguna atau hanya terbantu oleh jumlah kolom. Penelitian ini menganalisis Opcode Distribution pada 22.330 kontrak, 397 fitur, dan delapan label DASP. Lima konfigurasi multi-label diuji dengan 3×5 repeated cross-validation, dilanjutkan global ablation, size-controlled random opcode ablation, per-label degradation analysis, cumulative stability analysis, dan opcode-profile group-aware robustness checking. MultiOutput LightGBM mencapai Micro-F1 0.91396, Macro-F1 0.82464, dan Macro-PR-AUC 0.90146. Penghapusan seluruh Opcode Distribution menurunkan Macro-F1 menjadi 0.78745, sedangkan penghapusan subset acak berukuran sama menghasilkan Macro-F1 0.82404. Temuan ini menunjukkan Opcode Distribution sebagai representasi prediktif kolektif, bukan artefak jumlah fitur, tanpa membuktikan mekanisme kausal kerentanan.
Kata kunci: studi ablasi, Dataset DIVE, klasifikasi multi-label, distribusi opcode, deteksi kerentanan smart contract

1. Introduction

Smart contracts are executable programs deployed on blockchain infrastructures to automate agreements, transactions, and decentralized application logic. Their security has become a recurring research concern because vulnerable code can persist after deployment and may expose digital assets to irreversible exploitation. Recent reviews show that smart contract vulnerability detection has been studied through static analysis, symbolic execution, fuzzing, formal verification, machine learning, and deep learning, each with different trade-offs in scalability, interpretability, and vulnerability coverage [1]-[3]. Data-driven detection methods depend heavily on the quality of dataset labels and contract representations. Several studies have proposed datasets or detection models based on source code, bytecode, opcode sequences, graph structures, or multimodal features [4]-[15]. Nevertheless, many works still emphasize headline predictive scores without examining whether the representation that drives those scores is robust, balanced across labels, or inflated by dataset-specific artifacts. This issue is more visible in multi-label vulnerability detection because a single contract may contain several vulnerabilities and minority classes can be hidden by strong performance on frequent labels.

The DIVE Dataset provides a useful benchmark for this problem. It contains 22,330 verified Ethereum smart contracts deployed between 2016 and 2024, eight vulnerability labels aligned with the DASP taxonomy, and lifecycle-based features representing pre-deployment and post-deployment contract attributes [16]-[18]. DIVE also uses a multi-tool labeling pipeline based on Power-based voting and post-hoc validation [19]. Although DIVE improves dataset coverage and feature diversity, it does not by itself answer which feature groups carry predictive signal or whether dominant feature groups appear useful merely because they occupy a large part of the feature space. Prior smart contract vulnerability studies have explored bytecode-level learning, graph-based detection, language models, multimodal fusion, and benchmark construction [8]-[15]. These studies are valuable, but most do not isolate representation contribution under size-controlled feature removal. In particular, Opcode Distribution features are attractive because they summarize low-level Ethereum Virtual Machine instructions, yet they often dominate the feature space numerically. If removing them reduces performance, the loss may reflect real collective signal or simply the removal of many columns. This distinction matters for data science evaluation because feature-count artifacts can lead to misleading conclusions about representation usefulness.

Recent studies also broaden smart contract security research through tool evaluation, fuzzing, hybrid bytecode models, binary and multiclass pipelines, and multi-label dependency analysis [20]-[30]. Those directions strengthen vulnerability detection research, but they still leave a narrower evaluation problem: whether a large opcode-based representation remains useful after feature-count effects and exact opcode-profile overlap are controlled. This study addresses that methodological gap. Instead of proposing another classifier, it evaluates whether Opcode Distribution acts as a collective predictive representation for multi-label smart contract vulnerability detection. The experimental design combines model benchmarking, global feature-group ablation, size-controlled random opcode removal, per-label degradation analysis, cumulative cross-validation stability, and opcode-profile group-aware robustness checking. The contribution is therefore not a new detector architecture, but a controlled evaluation protocol that separates representation contribution from feature-count dominance and exact opcode-profile overlap.

Table 1. Position of this study against related work.

Study group	Main focus	Limitation for this study	Position of this study
Dataset construction	Large-scale smart contract labels and features [8]-[10], [16]-[19]	Usually describes dataset utility rather than controlled feature contribution	Uses DIVE to test representation contribution under ablation
Bytecode/ opcode detection	Learning from bytecode or opcode-level signals [11], [12]	Often reports performance without size-controlled feature removal	Tests whether opcode distribution remains useful beyond feature count
Graph and deep learning	GNN, language model, and fusion-based detectors [13]-[15]	Focuses on model architecture more than representation reliability	Uses stable tabular models as instruments for representation evaluation

Study group	Main focus	Limitation for this study	Position of this study
Dataset quality evaluation	Bias, leakage, and vulnerability dataset quality [31]	Rarely applied to smart contract feature groups	Adds group-aware robustness against exact opcode-profile overlap

2. Method

2.1. Dataset and Label Space

The study uses the processed DIVE dataset for multi-label smart contract vulnerability detection [16]. Each instance represents a verified Ethereum smart contract with tabular lifecycle features and eight binary vulnerability labels: Re-entrancy, Access Control, Arithmetic, Unchecked Return Values, DoS, Bad Randomness, Front Running, and Time Manipulation. The experiments use 22,330 contracts and 397 processed features. Since DIVE labels are derived from automated analysis tools and post-hoc validation, the results are interpreted as performance against the DIVE label space, not as manual proof of real-world exploitability. Table 2 summarizes the dataset. Table 3 shows that the label distribution is imbalanced. Access Control and Re-entrancy are frequent, while Bad Randomness and Front Running have positive ratios below 3%. This distribution motivates macro-averaged and per-label evaluation because micro-averaged metrics can be dominated by frequent labels [32], [33]. Figure 1 visualizes this imbalance and shows why macro-averaged and per-label metrics are needed.

Table 2. Dataset summary.

Characteristic	Value
Total smart contracts	22,330
Number of vulnerability labels	8
Final processed features	397
Task type	Multi-label smart contract vulnerability detection
Feature representation	Processed tabular lifecycle features
Evaluation design	3 × 5 repeated cross-validation

Table 3. Vulnerability label distribution.

Label	Positive count	Positive ratio
Access Control	16,723	0.7489
Reentrancy	11,400	0.5105
Arithmetic	9,542	0.4273
Time Manipulation	6,322	0.2831
Unchecked Return Values	5,911	0.2647
DoS	3,781	0.1693
Bad Randomness	634	0.0284
Front Running	606	0.0271

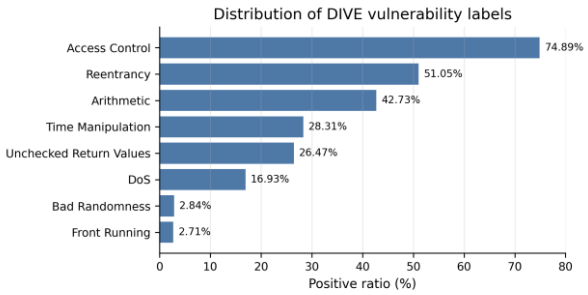

Figure 1. Distribution of positive vulnerability labels in the DIVE Dataset.

2.2. Feature Grouping

The processed features were grouped according to column names and lifecycle meaning. Structure features capture contract-level metadata and code metrics. Financial features capture value-related attributes. Runtime

features capture deployment and execution-related metadata. Opcode Category features aggregate opcode families, while Opcode Distribution features capture individual opcode-level frequency or distributional information. Features that did not clearly align with a single lifecycle group were retained as ungrouped auxiliary features. Opcode Distribution is the largest group, with 291 features or 73.30% of the processed feature space. This dominance creates the central methodological concern of the study. If full opcode removal causes a large performance drop, the drop must be tested against a feature-count explanation. Table 4 reports the feature groups used in the experiments.

Table 4. Feature groups used in this study.

Feature group	Number of features	Percentage
Structure	26	6.55%
Financial	8	2.02%
Runtime	14	3.53%
Opcode Category	20	5.04%
Opcode Distribution	291	73.30%
Ungrouped auxiliary features	38	9.57%

2.3. Experimental Design and Evaluation

Five multi-label learning configurations were evaluated: MultiOutput LightGBM, MultiOutput Random Forest, Classifier Chain Random Forest, MultiOutput ExtraTrees, and MultiOutput Logistic Regression. Preprocessing was fitted inside each training-fold pipeline: missing numeric values were imputed with the training-fold median, and categorical variables were one-hot encoded using categories learned only from the training partition before the transformation was applied to the held-out fold. MultiOutput learning trains one estimator per label, while Classifier Chain models label dependency by passing earlier label predictions to subsequent classifiers [32], [34]. Random Forest and LightGBM were retained as strong tabular baselines because they can model nonlinear relationships and feature interactions [35], [36]. Evaluation used 3 × 5 repeated cross-validation with repeat seeds 42, 7, and 21. Multi-label stratification was applied to preserve label distribution across folds [37]. Performance was measured with Micro-F1, Macro-F1, Weighted-F1, Hamming Loss, Micro-PR-AUC, and Macro-PR-AUC. Macro-F1 was emphasized because it gives equal weight to all labels and is less dominated by frequent classes. For label set Y with L labels, Macro-F1 is expressed in (1).

$$Macro - F1 = \frac{1}{L} \sum_{i=1}^{L} F1_i \qquad (1)$$

To preserve comparability across the baseline and ablation conditions, no GridSearchCV, randomized search, or fold-specific hyperparameter optimization was performed. The fixed configurations in Table 5 were established before repeated cross-validation and retained across every fold, repeat, and feature-removal scenario. Parameters not listed in the table followed the corresponding library defaults.

Table 5. Fixed model configurations used in the experiments.

Model	Fixed configuration
MultiOutput LightGBM	n_estimators = 300; learning_rate = 0.05; num_leaves = 31; max_depth = -1; subsample = 0.9; colsample_bytree = 0.9
MultiOutput Random Forest	n_estimators = 150
Classifier Chain Random Forest	Random Forest base estimator; n_estimators = 150; order = random
MultiOutput ExtraTrees	n_estimators = 150
MultiOutput Logistic Regression	solver = lbfgs; max_iter = 1000

Global feature-group ablation was performed with MultiOutput LightGBM, the best baseline model. Each feature group was removed one at a time and the model was re-evaluated using the same repeated cross-validation design. Group contribution was estimated as full-feature performance minus ablated performance; a larger drop indicates stronger predictive contribution under the tested configuration [38]. A size-controlled opcode ablation then separated representation contribution from feature-count effects. Within each of the 15 repeat-fold evaluations, three independent random subsets of 26 Opcode Distribution features were sampled

and removed, matching the size of the Structure group and yielding 45 evaluations. The subsets were resampled within each repeat-fold rather than fixed globally. Finally, opcode-profile group-aware robustness checking hashed Opcode Distribution profiles and enforced zero exact profile overlap between training and test groups.

3. Result and Discussion

3.1. Benchmark Performance

Table 6 presents the benchmark results. MultiOutput LightGBM achieved the strongest overall performance, with Micro-F1 of 0.91396, Macro-F1 of 0.82464, Weighted-F1 of 0.91139, Hamming Loss of 0.05210, and Macro-PR-AUC of 0.90146. The three tree-based alternatives produced close but lower Macro-F1 values. Logistic Regression performed substantially worse, indicating that the processed DIVE feature space contains nonlinear relationships that are not well captured by a simple linear decision boundary. The small difference between MultiOutput Random Forest and Classifier Chain Random Forest shows that the tested random-order chain did not provide a clear advantage over independent label-wise learning. This does not reject label dependency in DIVE, it only indicates that the tested chain configuration did not improve this representation.

Table 6. Benchmark results under 3 × 5 repeated cross-validation.

Model	Micro-F1	Macro-F1	Weighted-F1	Hamming Loss	Macro-PR-AUC	n
MultiOutput LightGBM	0.91396	0.82464	0.91139	0.05210	0.90146	15
MultiOutput Random Forest	0.90251	0.77515	0.89669	0.05818	0.88478	15
Classifier Chain Random Forest	0.90242	0.77413	0.89661	0.05822	0.88487	15
MultiOutput ExtraTrees	0.90151	0.77313	0.89563	0.05863	0.88688	15
MultiOutput Logistic Regression	0.48922	0.14778	0.32666	0.24429	0.30814	15

3.2. Feature-Group Ablation

Table 7 reports the global ablation results. Removing Opcode Distribution produced the largest degradation: Macro-F1 decreased from 0.82464 to 0.78745, a paired mean drop of 0.03719 with sample SD 0.00825 across the 15 repeat-fold comparisons. Hamming Loss increased from 0.05210 to 0.06020. The paired Wilcoxon signed-rank test gave $W = 0$ and $p < 0.001$, while 100,000 paired bootstrap resamples produced a 95% confidence interval of [0.0332, 0.0412] for the Macro-F1 drop. Figure 2 displays paired mean drops with sample-SD error bars. Removing Structure produced 0.00544 ± 0.00556, Financial 0.00123 ± 0.00427, Runtime -0.00052 ± 0.00330, and Opcode Category -0.00056 ± 0.00469. The small magnitudes and fold-level variation of these four conditions support a cautious interpretation: they may contain redundant or weak incremental information relative to the remaining features, rather than being irrelevant to smart contract security.

Table 7. LightGBM global feature-group ablation summary.

Ablation scenario	Micro-F1	Macro-F1	Hamming Loss	Macro-PR-AUC	Macro-F1 drop	n
Full Features	0.91396	0.82464	0.05210	0.90146	0.00000	15
Remove Opcode Distribution	0.90015	0.78745	0.06020	0.86845	0.03719	15
Remove Structure	0.91112	0.81920	0.05385	0.89605	0.00544	15
Remove Financial	0.91374	0.82341	0.05222	0.90005	0.00123	15
Remove Runtime	0.91382	0.82516	0.05219	0.90112	-0.00052	15
Remove Opcode Category	0.91437	0.82520	0.05185	0.90133	-0.00056	15

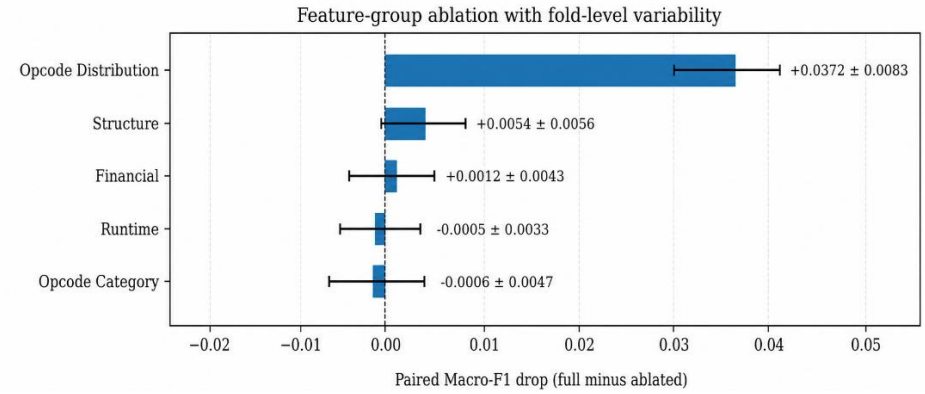

Figure 2. Mean paired Macro-F1 drop under feature-group ablation; error bars show sample SD across 15 matched repeat-fold comparisons.

3.3. Size-Controlled Opcode Ablation

The central concern with Opcode Distribution is its size: 291 features, or 73.30% of the feature matrix. Table 8 shows that removing 26 random opcode features produced pooled Macro-F1 of 0.82404 ± 0.00875 across 45 evaluations, nearly identical to the full-feature result of 0.82464 ± 0.00783 and clearly different from full Opcode Distribution removal at 0.78745 ± 0.00884. Figure 3 visualizes these scenario-level distributions. Because three subsets were sampled inside each repeat-fold, subset-selection variability was also isolated within the same data partitions. The pooled within-fold SD was 0.00368 and the mean within-fold range was 0.00630, both substantially smaller than the 0.03719 loss caused by complete Opcode Distribution removal. The result is therefore not driven by one favorable random subset. It supports Opcode Distribution as a distributed group-level representation whose useful signal is not reproduced by removing an arbitrary same-count subset.

Table 8. Full opcode removal versus size-controlled random opcode removal.

Scenario	Micro-F1	Macro-F1	Hamming Loss	Macro-PR-AUC	n
Full Features	0.91396	0.82464	0.05210	0.90146	15
Remove Full Opcode Distribution	0.90015	0.78745	0.06020	0.86845	15
Remove 26 Random Opcode Features	0.91376	0.82404	0.05222	0.89996	45

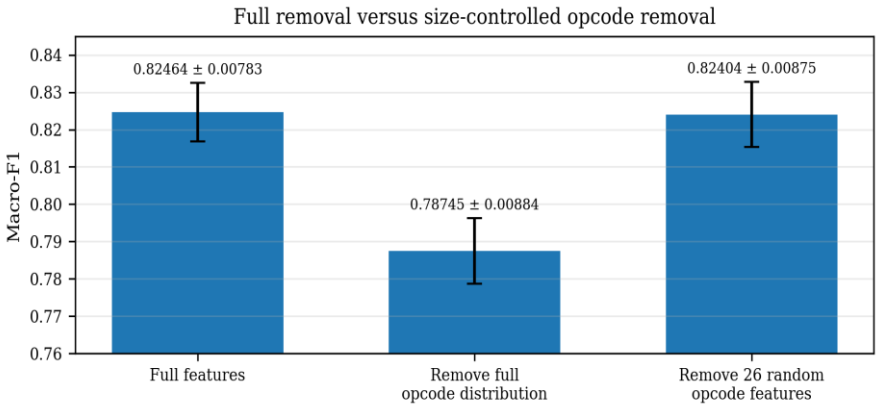

Figure 3. Macro-F1 mean ± sample SD for full features (n = 15), full Opcode Distribution removal (n = 15), and 26-feature random opcode removal (n = 45).

3.4. Per-Label Degradation and Robustness

Aggregate metrics can hide vulnerability-specific behavior. Table 9 shows that Opcode Distribution removal had the largest mean F1 effects on Bad Randomness (0.12848 ± 0.04065), Front Running (0.05943 ± 0.03379), Time Manipulation (0.04713 ± 0.00501), and DoS (0.02819 ± 0.00807). Access Control showed only 0.00093 ± 0.00117. Figure 4 reports the paired fold-level dispersion. The wording is intentionally based on mean degradation: not every label declined in every fold, and Front Running included one slightly negative fold-level difference. The pattern still shows that representation contribution is vulnerability-specific and that

minority-label behavior can be obscured by Micro-F1. The small Access Control effect does not imply that opcode information is generally irrelevant in DIVE and under the tested model, other retained features were sufficient to preserve most of its predictive performance.

Table 9. Per-label F1 drop caused by feature-group ablation.

Ablation scenario	Reentrancy	Access Control	Arithmetic	Unchecked Return Values	DoS	Bad Randomness	Front Running	Time Manipulation
Remove Opcode Distribution	0.0103	0.0009	0.0091	0.0140	0.0282	0.1285	0.0594	0.0471
Remove Structure	0.0015	0.0024	0.0009	0.0109	0.0040	0.0102	0.0127	0.0008
Remove Financial	-0.0009	-0.0001	-0.0007	0.0009	0.0043	-0.0045	0.0094	0.0014
Remove Runtime	-0.0007	0.0016	0.0002	-0.0013	-0.0023	-0.0017	-0.0005	0.0007
Remove Opcode Category	0.0002	0.0000	-0.0018	-0.0005	-0.0017	-0.0021	0.0011	0.0003

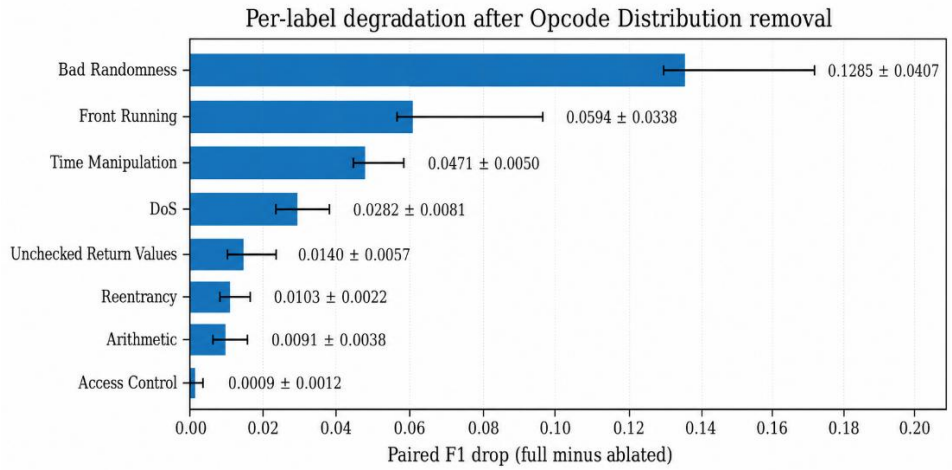

Figure 4. Mean per-label F1 drop after removing Opcode Distribution; error bars show sample SD across 15 matched repeat-fold comparisons.

The group-aware robustness check prevented exact Opcode Distribution profiles from crossing training and test partitions. Table 10 shows that performance remained close to the standard estimate: Macro-F1 was 0.82464 ± 0.00783 under standard repeated cross-validation (n = 15) and 0.82628 ± 0.00485 under group-aware splitting (n = 5), while Macro-PR-AUC was 0.90146 ± 0.00961 and 0.89873 ± 0.00328, respectively. Figure 5 therefore plots the two schemes separately with their own sample-SD error bars; because the observations are unpaired and their sample counts differ, no dispersion measure is attached directly to the mean-difference column. This result reduces the likelihood that the benchmark estimate is driven by exact duplicate opcode profiles, but the claim must remain narrow. The grouping does not identify semantic clones that differ in source code, bytecode organization, or control flow. It also does not eliminate regularities introduced by Solidity compiler versions, optimizer settings, standard libraries, or common deployment templates. A model may therefore learn implementation-family artifacts alongside vulnerability-related signal [31], [37].

Table 10. Standard repeated cross-validation versus opcode-profile group-aware robustness.

Metric	Standard CV mean	Standard CV std	Group-aware mean	Group-aware std	Difference
Micro-F1	0.91396	0.00116	0.91518	0.00093	+0.00122
Macro-F1	0.82464	0.00783	0.82628	0.00485	+0.00164
Weighted-F1	0.91139	0.00125	0.91272	0.00111	+0.00133
Hamming Loss	0.05210	0.00068	0.05108	0.00050	-0.00101
Micro-PR-AUC	0.97421	0.00072	0.97454	0.00075	+0.00034
Macro-PR-AUC	0.90146	0.00961	0.89873	0.00328	-0.00273

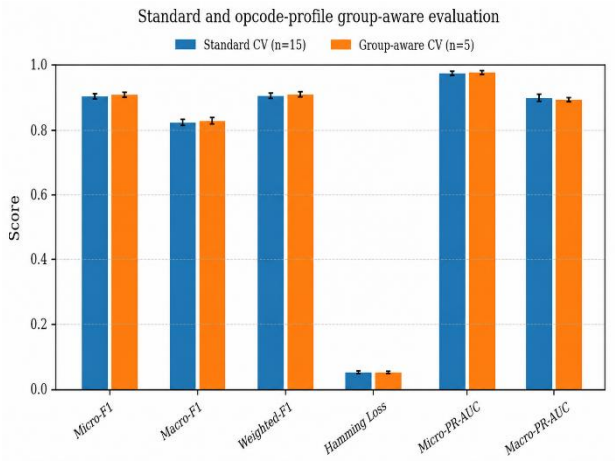

Figure 5. Standard repeated CV and opcode-profile group-aware performance shown separately as mean ± sample SD (n = 15 and n = 5, respectively).

3.5. Scientific Implications and Limits

The results have two implications for applied data science in software security. First, full-feature benchmark scores are insufficient for evaluating representation value: a dominant feature group should be tested with size-controlled removal and fold-level dispersion. Second, minority vulnerabilities require per-label analysis because their dependence on a representation can be hidden by micro-averaged performance. The findings remain bounded to DIVE, the tested model family, and the specified removal operations. Ablation identifies predictive contribution, not causal vulnerability mechanisms. Opcode Distribution also omits instruction order, control-flow paths, transaction context, and cross-contract interactions, while DIVE labels inherit uncertainty from automated tools rather than full manual audits. The safest conclusion is that Opcode Distribution is a collective predictive representation for DIVE-based multi-label detection, not a complete or causal explanation of exploitability.

4. Conclusion and Future Research

This study evaluated Opcode Distribution in multi-label smart contract vulnerability detection using the DIVE Dataset. MultiOutput LightGBM model produced the strongest baseline, with Micro-F1 of 0.91396 and Macro-F1 of 0.82464. Complete Opcode Distribution removal caused the largest Macro-F1 decrease, whereas removing a 26-feature random opcode subset did not reproduce the loss. Per-label analysis showed the largest mean effects for Bad Randomness, Front Running, Time Manipulation, and DoS, and exact-profile group-aware splitting produced estimates close to standard repeated cross-validation. These findings support Opcode Distribution as a collective predictive representation rather than a feature-count artifact, without establishing causality. Future validation should harmonize vulnerability taxonomies between DIVE and independent datasets; remove exact duplicates and group near-duplicate families using normalized source code, bytecode, control-flow, and compiler metadata; train on DIVE and test externally without retraining or threshold adjustment, then reverse the direction; and construct holdouts by Solidity version, optimizer setting, common template, and deployment period. Targeted comparison with independent audit evidence would further assess label reliability and external validity.

Acknowledgements

The authors thank the maintainers of the DIVE Dataset and DIVE Framework for making the dataset and supporting materials publicly available for research use.

Conflict of Interest

The authors declare that there is no conflict of interest related to this research or manuscript.

Data Availability

This study uses the publicly available DIVE Dataset, which is not redistributed in this article. An anonymized review repository containing the experimental scripts, feature-group mappings, derived result tables, and figure-generation materials has been supplied through the journal submission system. The permanent public repository link will be restored after peer review.

Declaration of AI-Assisted Writing

AI-assisted writing tools were used only for language refinement, readability improvement, and formatting. They were not used to generate research data, run experiments, perform statistical analysis, create figures, interpret results, or draw scientific conclusions. All assisted text was reviewed and revised by the authors, who take full responsibility for the manuscript.

References

[1] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Systematic review of security vulnerabilities in Ethereum blockchain smart contract," IEEE Access, vol. 10, pp. 6605-6621, 2022, doi: 10.1109/ACCESS.2021.3140091.
[2] S. J. Alsunaidi, H. Aljamaan, and M. Hammoudeh, "Leveraging machine learning models to improve smart contract security: A survey of vulnerabilities and detection methods," ACM Computing Surveys, vol. 58, no. 6, Art. no. 151, pp. 1-37, 2025, doi: 10.1145/3772367.
[3] F. Jiang et al., "Enhancing smart-contract security through machine learning: A survey of approaches and techniques," Electronics, vol. 12, no. 9, p. 2046, 2023, doi: 10.3390/electronics12092046.
[4] Z. Zheng et al., "DAppSCAN: Building large-scale datasets for smart contract weaknesses in DApp projects," IEEE Transactions on Software Engineering, vol. 50, pp. 1360-1373, 2024, doi: 10.1109/TSE.2024.3383422.
[5] C. S. Yashavant, S. Kumar, and A. Karkare, "ScrawlD: A dataset of real world Ethereum smart contracts labelled with vulnerabilities," arXiv, 2022, doi: 10.48550/arXiv.2202.11409.
[6] G. Ibba et al., "A curated Solidity smart contracts repository of metrics and vulnerability," in Proceedings of the 20th International Conference on Predictive Models and Data Analytics in Software Engineering, 2024, pp. 32-41, doi: 10.1145/3663533.3664039.
[7] S. HajiHosseinKhani, A. H. Lashkari, and A. M. Oskui, "Unveiling smart contracts vulnerabilities: Toward profiling smart contracts vulnerabilities using enhanced genetic algorithm and generating benchmark dataset," Blockchain: Research and Applications, vol. 6, no. 2, Art. no. 100253, 2025, doi: 10.1016/j.bcra.2024.100253.
[8] P. Qian, Z. Liu, Y. Yin, and Q. He, "Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode," in Proceedings of the ACM Web Conference 2023, 2023, pp. 2220-2229, doi: 10.1145/3543507.3583367.
[9] W. Deng et al., "Smart contract vulnerability detection based on deep learning and multimodal decision fusion," Sensors, vol. 23, no. 16, p. 7246, 2023, doi: 10.3390/s23167246.
[10] F. Luo et al., "SCVHunter: Smart contract vulnerability detection based on heterogeneous graph attention network," in Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1-13, doi: 10.1145/3597503.3639213.
[11] Y. Wang, X. Zhao, L. He, Z. Zhen, and H. Chen, "ContractGNN: Ethereum smart contract vulnerability detection based on vulnerability sub-graphs and graph neural networks," IEEE Transactions on Network Science and Engineering, vol. 11, pp. 6382-6395, 2024, doi: 10.1109/TNSE.2024.3470788.
[12] X. Sun et al., "ASSBert: Active and semi-supervised BERT for smart contract vulnerability detection," Journal of Information Security and Applications, vol. 73, p. 103423, 2023, doi: 10.1016/j.jisa.2023.103423.
[13] B. Le Hung et al., "Contextual language model and transfer learning for reentrancy vulnerability detection in smart contracts," in Proceedings of the 12th International Symposium on Information and Communication Technology, 2023, pp. 739-745, doi: 10.1145/3628797.3628945.
[14] V. K. Jain and M. Tripathi, "An integrated deep learning model for Ethereum smart contract vulnerability detection," International Journal of Information Security, vol. 23, pp. 557-575, 2024, doi: 10.1007/s10207-023-00752-5.
[15] H. Lakadawala, K. Dzigbede, and Y. Chen, "Detecting reentrancy vulnerability in smart contracts using graph convolution networks," in 2024 IEEE 21st Consumer Communications & Networking Conference, 2024, pp. 188-193, doi: 10.1109/CCNC51664.2024.10454763.
[16] S. J. Alsunaidi, H. Aljamaan, and M. Hammoudeh, "DIVE: A multi-label smart contract vulnerability dataset," Scientific Data, vol. 13, p. 664, 2026, doi: 10.1038/s41597-026-07025-5.
[17] S. Alsunaidi, H. Aljamaan, and M. Hammoudeh, "DIVE: A multi-label smart contract vulnerability dataset," Zenodo, 2026, doi: 10.5281/zenodo.18519253.
[18] DIVE Framework, "DIVE, version v2.0.0," Zenodo, 2026, doi: 10.5281/zenodo.18779606.
[19] S. J. Alsunaidi, H. Aljamaan, and M. Hammoudeh, "MultiTagging: A vulnerable smart contract labeling and evaluation framework," Electronics, vol. 13, no. 23, p. 4616, 2024, doi: 10.3390/electronics13234616.
[20] B. Lashkari and P. Musilek, "Evaluation of smart contract vulnerability analysis tools: A domain-specific perspective," Information, vol. 14, no. 10, p. 533, 2023, doi: 10.3390/info14100533.

[21] R. Yu, J. Shu, D. Yan, and X. Jia, "ReDetect: Reentrancy vulnerability detection in smart contracts with high accuracy," in 2021 17th International Conference on Mobility, Sensing and Networking, 2021, pp. 412-419, doi: 10.1109/MSN53354.2021.00069.
[22] Z. Zheng et al., "Turn the rudder: A beacon of reentrancy detection for smart contracts on Ethereum," in 2023 IEEE/ACM 45th International Conference on Software Engineering, 2023, pp. 295-306, doi: 10.1109/ICSE48619.2023.00036.
[23] N. Ivanov et al., "Security threat mitigation for smart contracts: A comprehensive survey," ACM Computing Surveys, vol. 55, no. 14s, Art. no. 326, pp. 1-37, 2023, doi: 10.1145/3593293.
[24] H. Zhou, A. Milani Fard, and A. Makanju, "The state of Ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support," Journal of Cybersecurity and Privacy, vol. 2, no. 2, pp. 358-378, 2022, doi: 10.3390/jcp2020019.
[25] Z. Liu et al., "Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting," IEEE Transactions on Information Forensics and Security, vol. 18, pp. 1237-1251, 2023, doi: 10.1109/TIFS.2023.3237370.
[26] L. Zhang et al., "SPCBIG-EC: A robust serial hybrid model for smart contract vulnerability detection," Sensors, vol. 22, no. 12, p. 4621, 2022, doi: 10.3390/s22124621.
[27] Z. Yang, W. Zhu, and M. Yu, "Improvement and optimization of vulnerability detection methods for Ethereum smart contracts," IEEE Access, vol. 11, pp. 78207-78223, 2023, doi: 10.1109/ACCESS.2023.3298672.
[28] M. Eshghie, C. Artho, and D. Gurov, "Dynamic vulnerability detection on smart contracts using machine learning," in Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, 2021, pp. 305-312, doi: 10.1145/3463274.3463348.
[29] A. Mezina and A. Ometov, "Detecting smart contract vulnerabilities with combined binary and multiclass classification," Cryptography, vol. 7, no. 3, p. 34, 2023, doi: 10.3390/cryptography7030034.
[30] S. Rawlekar, S. Bhatnagar, V. P. Srinivasulu, and N. Ahuja, "Improving multi-label recognition using class co-occurrence probabilities," in Pattern Recognition: 27th International Conference, ICPR 2024, Proceedings, vol. 15310, pp. 424-439, Springer, 2025, doi: 10.1007/978-3-031-78192-6_28.
[31] R. Croft, M. A. Babar, and M. M. Kholoosi, "Data quality for software vulnerability datasets," in 2023 IEEE/ACM International Conference on Software Engineering, 2023, pp. 121-133, doi: 10.1109/ICSE48619.2023.00022.
[32] S. Huang, W. Hu, B. Lu, Q. Fan, X. Xu, X. Zhou, and H. Yan, "Application of label correlation in multi-label classification: A survey," Applied Sciences, vol. 14, no. 19, p. 9034, 2024, doi: 10.3390/app14199034.
[33] A. N. Tarekegn, M. Giacobini, and K. Michalak, "A review of methods for imbalanced multi-label classification," Pattern Recognition, vol. 118, p. 107965, 2021, doi: 10.1016/j.patcog.2021.107965.
[34] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains: A review and perspectives," Journal of Artificial Intelligence Research, vol. 70, pp. 683-718, 2021, doi: 10.1613/jair.1.12376.
[35] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in Advances in Neural Information Processing Systems, vol. 35, pp. 507-520, 2022.
[36] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 6, pp. 7499-7519, 2024, doi: 10.1109/TNNLS.2022.3229161.
[37] H. Tiittanen, L. Holm, and P. Törönen, "Novel split quality measures for stratified multilabel cross validation with application to large and sparse gene ontology datasets," Applied Computing and Intelligence, vol. 2, no. 1, pp. 49-62, 2022, doi: 10.3934/aci.2022003.
[38] I. Covert, S. Lundberg, and S.-I. Lee, "Explaining by removing: A unified framework for model explanation," Journal of Machine Learning Research, vol. 22, no. 209, pp. 1-90, 2021.