Majestic Journal of Information Technology
Volume 1, 2026, **pp.** 1-17, **Article ID:** CM2601102003

Research Article

A Reproducibility Audit and Reconstruction of a Synthetic-Schema Multimodal Gated-Fusion Pipeline for Function-Level Software Vulnerability Analysis

Nasir Uddin Khan[1, *], ✉

[1]*Department of Emaan Institute of Management Sciences, Karachi, Pakistan*

Article History

Received: 16 May, 2026 Revised: 08 September, 2026 Accepted: 14 September, 2026 Published: 25 September, 2026

Abstract:

Introduction: Automated source-code vulnerability analysis has gained attention because it can support security assessment at scale; however, reliable evaluation requires transparent datasets, leakage-resistant validation and independently executable computational procedures.

Methodology: This study presents a documented reconstruction and audit of a lightweight multimodal gated-fusion pipeline intended for function-level vulnerability classification. Because authenticated Big-Vul records could not be obtained during execution, the implemented analysis used a schema-compatible synthetic surrogate rather than real vulnerability data. The reconstructed pipeline combines word-level TF-IDF-SVD representations, character-level TF-IDF-SVD representations and six structural indicators through a sample-specific fusion mechanism.

Results: The executed synthetic evaluation produced a mean macro F1 score of 0.8519; however, this result describes behaviour within the generated surrogate environment and does not establish real-world vulnerability-detection performance. The audit identified preprocessing leakage, template-lineage overlap, hard-coded comparison entries and differences between the intended architecture and the executed implementation.

Conclusion: The study therefore contributes a methodological audit framework and specifies requirements for future validation using authenticated datasets, grouped evaluation strategies and independently generated experimental comparisons.

Keywords: Software vulnerability analysis, multimodal learning, gated fusion, reproducibility audit, synthetic surrogate evaluation, source-code analysis, implementation verification.

1. INTRODUCTION

The potential for compromise of the system, exposure of data, and downstream supply-chain risk are all risks to consider when talking about software vulnerabilities [1]. The goal of function level vulnerability classification is to see if a function, that is available in the source code, shows security-relevant behaviour before deployment or maintenance [2]. To reliably assess, it is necessary to have genuine datasets of vulnerabilities that maintain realistic project structure, historical commits, duplicate relations and naturally occurring coding patterns. If these are not present, model performance can be due to artefacts created in building the dataset, instead of transferable understanding of vulnerability. Common Vulnerabilities and Exposures (CVEs) are mapped with C/C++ source-code changes in hundreds of open-source projects by Big-Vul [3]. But the size of the code base by itself is no guarantee of realistic evaluation: vulnerability labels can carry a lot of noise, functions from related projects or commits may be split up between the training and test partitions, and the presence of a severe class imbalance can hide the poor detection of vulnerable code.

Empirical evaluations in recent years have shown that impact of duplicate artefacts, artificial class distributions and token-level shortcuts can significantly reduce the performance of vulnerability detection. In this regard, a credible study should contain not just information on the architecture, but also

[*]Address correspondence to this author at Department of Emaan Institute of Management Sciences, Karachi, Pakistan E-mail: rector@emaan.edu.pk

© 2026 Copyright by the Author.
Licensed as an open access article using a CC BY 4.0 license.

Cite as: N.U. Khan, "A reproducibility audit and reconstruction of a synthetic-schema multimodal gated-fusion pipeline for function-level software vulnerability analysis," *Maj. J. Inf. Technol.,* vol. 1, pp.1–17, 2026, Article ID: CM2601102003.

information on the data source, how the data was cleaned, how it was grouped, how it was split, random seeds, the loss function, and the source of each of the metrics reported. Transparent reporting of datasets, preprocessing techniques, evaluation protocols and computational artefacts are critical to the evaluation of the reliability of empirical claims in classification research where security issues are paramount [4].

At the same time, there has been significant progress in representation learning. To achieve transferable representations of natural and programming languages, CodeBERT was proposed [5] and GraphCodeBERT was proposed to add data-flow objectives during code pretraining [6]. Following this work on interpretable program-dependence representations as IVDetect [7] and transformer-based localisation as LineVul [8] have done. These techniques illustrate the importance of complementing these contextual, lexical and structural features, but pretrained or graph-based methods can be memory intensive, require a lot of preprocessing and training efforts.

The original implementation specification was for a more sophisticated vulnerability classification framework using CodeBERT representations, bidirectional recurrent encoding, focal loss, label smoothing, grouped validation, calibration analysis and Integrated Gradients. The executed pipeline was a significantly lighter model, consisting of two TF-IDF-SVD lexical branches, six hand-crafted structural counts, a trainable linear classifier, a softmax gate, four-fold StratifiedKFold validation, weighted softmax cross-entropy and finite-difference attribution. Data was not acquired using Kaggle, but instead a surrogate of 36 columns was created for the data from the code. The notebook also includes hard-coded baseline entries recorded in the implementation artifact for CNN, BiLSTM and Transformer models and hard-coded ablation, PR-AUC and ECE entries - which is not an experimentally measured comparison for these models.

The study critically audits the implemented computational workflow, evaluates the evidential limitations of the synthetic reconstruction, contrasts the intended specification with the executed implementation, and establishes methodological requirements for future validation using authentic vulnerability datasets.

Based on this, four research questions have been proposed:

RQ1: What computational procedures, feature representations and evaluation mechanisms were actually implemented in the reconstructed vulnerability analysis pipeline?

RQ2: What methodological differences exist between the original implementation specification and the executed computational workflow?

RQ3: What reproducibility, validation and data-related risks affect interpretation of the executed synthetic evaluation?

RQ4: Based on methodological audit and synthesis rather than a new experiment, what controls are required to establish a leakage-resistant validation framework using authentic vulnerability datasets?

The contribution has three components. First, the study does not introduce or validate a new vulnerability-classification architecture; it reconstructs and documents the lightweight multimodal gated-fusion implementation that was actually executed, distinguishing implemented components from those proposed but not realised. Second, it audits the available implementation by separating directly computed outputs from hard-coded or illustrative entries and does not claim complete independent reproduction from the currently available materials. Third, it specifies a leakage-resistant procedure for future use of authenticated Big-Vul records, including commit- or project-grouped partitioning, independently trained comparisons and ablations, fold-specific preprocessing, probability calibration and defensible uncertainty estimates.

2. LITERATURE REVIEW

The early learning based vulnerability detectors typically encoded source code as sequences of tokens or graphs. This was followed by additional research that demonstrated the potential for models to be built on surface properties instead of vulnerability mechanisms (project conventions or patch artefacts). The control of unrealistic dataset artefacts led to a performance decrease in re-evaluation work [4] and larger empirical studies were made possible with Big-Vul [3]. The results urge the need for a clear separation between within-distribution performance and generalisation to unseen projects, commits or time periods, which is relevant to security.

Contextual representations that are hard to obtain using hand-crafted lexical features can be obtained using pretrained code language models. The CodeBERT [5] and GraphCodeBERT [6] pretraining models are designed for natural and programming language respectively, whereas the latter also learns about data-flow objectives to facilitate structural code understanding. These are typically fine-tuned into function classifiers or as embedding backbones for vulnerability tasks. They have pros such as contextual token representations and transfer learning, and cons such as the use of large amounts of GPU-memory and sensitivity to training-data quality, as well as token truncation.

Interpretability and localisation are also at the heart of the concerns. IVDet [7] models vulnerable statements and their dependency context to produce fine-grained interpretations, and LineVul [8] uses function-level prediction along with line-level ranking to aid in actionable code review. In the finite difference description of the notebook, the prediction sensitivity is much less expressive, instead of source tokens or statements. It can be used to model as a visualisation, but not to find the line of the source code that is responsible for a prediction.

Data quality is now a major methodological issue. DiverseVul expanded the scope of the project and specifically studied label noise [9]. Focusing on the need to properly dedup, split chronologies, PrimeVul also demonstrated that poor controls can significantly overestimate the actual performance of vulnerability detection in the real world [10]. These are reasons to consolidate related functions and to keep the order of

the commits when it has been recorded (and not randomised by row-level partitioning) in the current audit.

Program-analysis-informed methods try to incorporate structural information more directly. DeepDFA makes use of the concepts of data-flow-analysis and graph-learning to increase efficiency of multiple neural comparators [11]. Other cross-project approaches that integrate expert and semantic knowledge, such as CSVD-TF, have been proposed in transfer learning [12]; and a supervised contrastive learning method using GraphCodeBERT has been proposed to enhance the robustness in similar safe and vulnerable functions [13]. Although these studies justify the examination of multimodal fusion, the computation of every branch and its independent evaluation is necessary. The provision of a semantic, contextual and structural branch is not enough to consider that they are contributing to the computation of multimodal fusion.

Class imbalance and data sparseness are stubbornly the challenges. Minority-class augmentation with an attention to preserving vulnerable signal or sense has been investigated to improve minority-class coverage without discarding the underlying vulnerability signal [14]. However, the performance may significantly suffer if the detectors are tested in the realistic codebase distribution, which also includes a large number of functions that are not vulnerable. Empirical investigation conducted on the imbalance remedies revealed that there is no best remedy for all precision, recall and F1 [15]. Hence, a loss function and a sampling strategy should be chosen beforehand according to a set of pre-determined evaluation goals and the relative significance of the errors made on the classes.

Robustness and explanation quality are also ongoing research topics. Spurious correlations can be mitigated through VISION's counterfactual augmentation and better interpretability [16], and dependency-graph information can be added to optimised CodeBERT pipelines along with imbalance-aware loss design [17]. These methods are not used in the run notebook and are only described in the context of what might be evaluated in the future.

Probability calibration also warrants methodological attention in vulnerability classification. A classifier may assign correct labels while misestimating confidence, which affects thresholding and analyst workload. Calibration should be assessed using reliability diagrams, proper scoring rules and explicitly calculated calibration errors [18]. Post-hoc methods such as temperature scaling should be fitted on a held-out calibration or validation set while preserving an untouched test set for final evaluation [19]. The notebook contains ECE constants but does not compute ECE from model-predicted probabilities; calibration is therefore treated as an unmet validation requirement rather than an achieved result.

3. METHODOLOGY

3.1. Study Design and Evidential Scope

This study is a computational reconstruction and the differentiation of the two evidence layers. The first includes quantities which are computed directly in execution such as synthetic data counts, cross validation macro F1, precision and recall, seed summaries, final fold predictions, finite difference sensitivities and the truncation diagnostic. The second includes literal constants hard-coded in the hard-coded baseline entries in the implementation artifact, hard-coded PR-AUC/ECE entries, and the majority of ablation entries, which remain only for audit purposes for the sake of completeness of the evidence, but are not considered experimental evidence. Reconstructed refers to a component that could be recovered from the artefact that was found; audited refers to a component that was compared to the notebook and saved outputs; and reproducible refers to a component that can be executed independently from the materials found. The present study sets the ground for reconstruction and audit and not for full independent reproducibility.

3.2. Data Acquisition Audit

The notebook attempts to use the Kaggle command-line client to download the MSR data archive, but the command or MSR_data_cleaned.csv was unavailable. The exception handler therefore generated a synthetic 36-column table designed to resemble the reported Big-Vul schema. The generator created 188,636 base records, appended 200 exact copies, and returned to 188,636 unique records after duplicate removal. The surrogate does not preserve authentic Big-Vul vulnerability semantics, project relationships, commit history, duplicate structure or natural class distribution. For context, published Big-Vul statistics report 10,900 vulnerable and 177,736 non-vulnerable functions (188,636 total; 5.78% vulnerable), whereas the executed post-noise surrogate contains 46,173 vulnerable and 142,463 safe records (188,636 total; 24.48% vulnerable). These counts refer to distinct datasets and must not be combined. The identical total row count is therefore a schema-level coincidence rather than evidence of dataset equivalence.

3.3. Synthetic Generation Process

The surrogate generator uses six vulnerable/safe C-like template families: strcpy, sprintf_system, gets, use_after_free, integer_overflow and unchecked_printf. These identifiers denote strcpy misuse, combined sprintf/system-call patterns, gets misuse, use-after-free patterns, integer-overflow patterns and unchecked printf formatting, respectively. Safe counterparts use bounded or validated constructs such as strncpy, snprintf, fgets, explicit format strings, controlled command execution, memory checks and range checks. The family selector assigns each template family a probability of 1/6; this is distinct from the vulnerable/safe class prior, which is governed by separate class-generation logic. Because the exact class-assignment implementation is not preserved, the observed post-noise vulnerable proportion of 24.48% is reported rather than inferred from the family probability. Eight synthetic project identifiers are used: linux-kernel, openssl, ffmpeg, httpd, tcpdump, libxml2, php-src and chromium. The surrogate also contains six synthetic CWE identifiers: CWE-119, CWE-120, CWE-78, CWE-134, CWE-401 and CWE-476; however, the exact pairwise CWE-template mapping code is unavailable and is not claimed to be independently reconstructible.

Only some generation steps can be recovered deterministically from the available artefacts. Project labels are cycled across the eight predefined projects; function names follow the handle_{record_id}{suffix} pattern with suffixes cycling over req, msg, pkt, data and buf; and the label-noise indices use fixed positions. Distractor selection/insertion and later sampling steps are pseudo-random or insufficiently specified. No exact random seed, random-number-generator configuration, random-library version or complete sampling implementation is preserved. The notebook inverts the labels of the first 18,863 of 188,636 rows (10%) and stores a label_noise indicator. The saved post-noise output contains 142,463 safe and 46,173 vulnerable records, but the pre-noise class totals and the safe/vulnerable composition of the inverted subset were not saved. Consequently, exact regeneration of the surrogate is not claimed. Unavailable generation details include complete template and distractor definitions, insertion probabilities and count distributions, exact naming/project-assignment code, CWE-template mapping, random seeds and generator configuration, generation order, label-noise order and duplicate-generation logic.

3.3.1. Risk of Generator-Induced Shortcut Learning

Because the synthetic dataset was created from predefined vulnerable and safe template families, the generated labels may be strongly associated with artificial lexical and structural patterns. Several model inputs, including counts of risky API calls such as strcpy, gets, sprintf and system-related operations, are also embedded in the generation rules. The classifier may therefore learn generator-specific cues rather than transferable vulnerability characteristics. This generator-induced shortcut-learning risk is distinct from conventional preprocessing leakage and is considered when interpreting the within-surrogate results.

3.4. Cleaning and Pre-Processing

Missing values in predictor metadata are handled separately from the target label. Categorical or text metadata fields such as project, CWE ID, func_after, func_before, func_name, function_suffix, template_family and distractor_type are filled using mode imputation, while numeric fields such as Id and distractor_count are handled by the notebook's generic numeric-imputation routine. The generic imputation routine appears to include the target variable vul; however, the executed surrogate already contains complete target labels. In a corrected implementation, vul must be explicitly excluded from all predictor-imputation procedures, and any row with a missing target should be removed or flagged rather than imputed. The all-missing Known Exploits field is replaced with zero. For feature extraction, the executed pipeline uses the generated source-code representation stored in func_after.

3.5. Modelling Subsample

For computational efficiency, the notebook selects a random subsample of 12,000 records from the 188,636 unique post-noise surrogate records. The resulting modelling sample contains 9,063 safe and 2,937 vulnerable records, corresponding to a safe-to-vulnerable ratio of approximately 3.09:1. No intermediate filtering stage is documented between the 188,636-record surrogate and the 12,000-record modelling sample, so the workflow is described directly as subsampling from the full post-noise surrogate. The subsampling step precedes the cross-validation random_state=42 setting, and no fixed seed for the subsample is documented. Related observations are not grouped before cross-validation; the consequences of this partitioning choice are assessed in Section 3.9.1.

3.6. Multimodal Representation

The word-level lexical branch applies TF-IDF to unigrams and bigrams with a maximum vocabulary of 400 features and then reduces the sparse vector to 32 dimensions using truncated singular-value decomposition. The character-level branch uses 3- to 5-character n-grams with the same 400-feature cap and 32-dimensional SVD reduction. Character n-grams can capture local lexical patterns, operator substrings, identifier fragments and formatting regularities, but they are not contextual transformer encoders. These representations should therefore not be interpreted as equivalent alternatives to pretrained code-language models such as CodeBERT or GraphCodeBERT.

The structural branch contains six indicators: code length, opening-brace count, closing-brace count, semicolon count, risky-operation count and comparatively safer-operation count. Code length is measured as the number of characters in the raw source-code string, while braces and semicolons are counted directly. Risky-operation counts use explicit API or pattern indicators such as strcpy, sprintf, unchecked printf usage, system () and unsafe memory-management patterns such as use-after-free. Safer-operation counts use bounded or validated patterns such as strncpy, snprintf, fgets, explicit or validated format strings, controlled command-execution alternatives and memory-management checks. Matching is case-sensitive and operates on raw code without parsing, so comments and string literals remain present. Importantly, several of these risky/safe cues are also used to construct the synthetic templates. This represents generator-induced shortcut learning rather than conventional preprocessing leakage: the same lexical and structural cues used to construct the synthetic labels are also available to the classifier as predictive features.

$$TFIDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right) \tag{1}$$

$$X \approx U_k \Sigma_k V_k^T \tag{2}$$

3.7. Adaptive Gated Fusion

Fig. (**1**) summarises the executed architecture and Fig. (**2**) details the fusion operation. The structural vector is projected to the shared 32-dimensional space: $p_i = W_p R_i + b_p$. A three-element gate is then calculated as:

$$g_i = softmax\left(W_g \begin{bmatrix} h_i^s \\ h_i^c \\ r_i \end{bmatrix} + b_g\right),$$

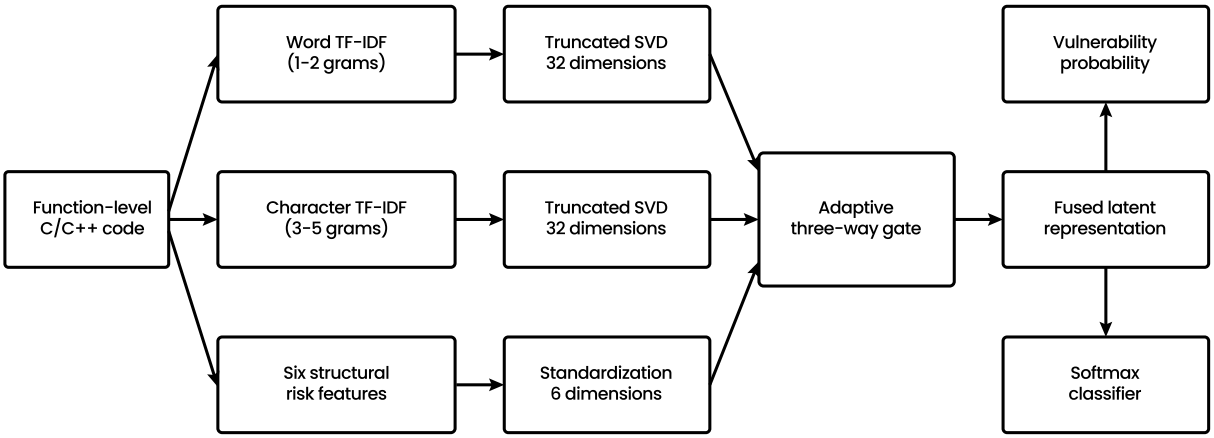


Fig. (1). Executed lightweight multimodal pipeline reconstructed from the synthetic surrogate implementation.

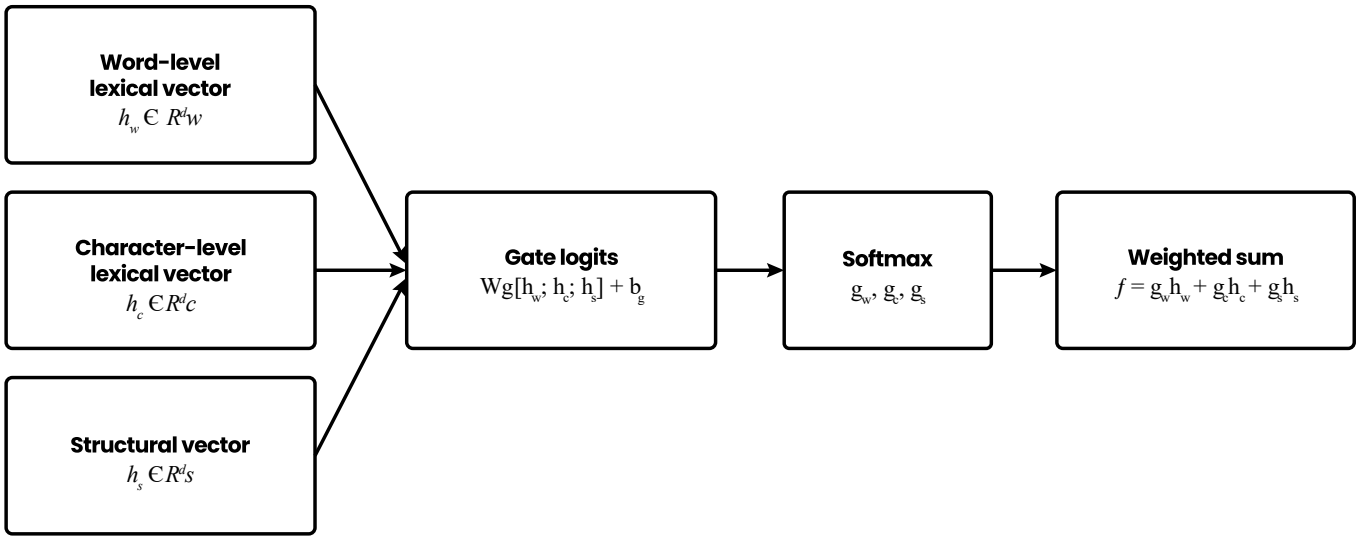


Fig. (2). Three-way gated fusion mechanism implemented in the reconstructed synthetic evaluation pipeline.

where the gate components g_i^s, g_i^c and g_i^r are non-negative and sum to one. The fused representation is $f_i = g_i^s h_i^s + g_i^c h_i^c + g_i^r p_i$. Finally, class probabilities are produced by $softmax(W_o f_i + b_o)$. The main difference in architecture that can be reconstructed from the executed notebook is this sample-specific weighting; the gate decides how much weight to give to each branch of the network's prediction.

$$p_i = W_p R_i + b_p \tag{3}$$

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4}$$

$$g_i = softmax\left(W_g \begin{bmatrix} h_i^s \\ h_i^c \\ r_i \end{bmatrix} + b_g\right) \tag{5}$$

$$g_i^s + g_i^c + g_i^r = 1 \tag{6}$$

$$f_i = g_i^s h_i^s + g_i^c h_i^c + g_i^r p_i \tag{7}$$

$$\hat{y}_i = softmax(W_o f_i + b_o) \tag{8}$$

3.8. Optimisation

The parameters are randomly initialised from a Gaussian distribution and optimised by manually implementing full-batch gradient descent for 250 epochs with a learning rate of 0.25. The safe class weight is 1.0 and the vulnerable class weight is $9{,}063/2{,}937 = 3.0858$. Weighted softmax cross-entropy is implemented. In contrast to the outline, the focal loss and label smoothing are not present. The executed notebook contains no minibatching, dropout, weight decay, early stopping, independent test set or separate validation set for

hyperparameter selection. The omissions make the procedure more easily executed, but can also make it more prone to overfitting and less powerful in terms of the resulting performance estimates.

$$w_{vulnerable} = \frac{9063}{2937} = 3.0858 \quad (9)$$

$$L = -\sum_{i=1}^{N} w_{yi} \log\big(p(y_i|x_i)\big) \quad (10)$$

3.9. Validation Protocol

The executed notebook uses StratifiedKFold with four folds and three random seeds.The reported metrics are macro F1, macro precision and macro recall. The protocol is documented as part of the implementation audit rather than presented as a leakage-resistant estimate of generalisation. The specific leakage channels created by pre-fitted preprocessing and ungrouped synthetic lineages are assessed in Section 3.9.1.

$$Macro - F1 = \frac{1}{C}\sum_{c=1}^{C} F1_c \quad (11)$$

where

$$F1 = \frac{2PR}{P+R} \quad (12)$$

Under class imbalance, macro averaging gives equal weight to each class and is more informative than accuracy alone. The executed protocol does not use StratifiedGroupKFold, and no confidence interval is computed by independent resampling. Because TF-IDF, SVD and feature standardisation were fitted before fold separation and template lineages were not grouped, the reported cross-validation scores should therefore be understood as leakage-prone, within-surrogate estimates that may be optimistically biased and should not be interpreted as unbiased out-of-sample performance estimates.

3.9.1. Leakage Assessment of the Executed Validation Procedure

Two leakage channels were identified in the executed evaluation procedure. Preprocessing leakage arises because TF-IDF transformation, SVD dimensionality reduction and feature standardisation were fitted before cross-validation rather than independently within each training fold. Synthetic-lineage leakage arises because related template-generated functions can appear in different folds. The latter is distinct from generator-induced shortcut learning (Section 3.3.1), which concerns label-construction cues being directly available as predictive features.

3.10. Diagnostic Analyses

For the confusion matrix, latent attribution and truncation diagnostic, the final fold is retained. Vulnerable-class probability sensitivity is approximated by a finite-difference perturbation of each word-level lexical SVD component. This output should be interpreted only as a local latent-space sensitivity diagnostic; it is not a semantic or contextual explanation and is not Integrated Gradients. The truncation diagnostic retains 100%, 70%, 50%, 30% and 15% of each function from the beginning, recomputes all branches and then recalculates macro F1, thereby measuring sensitivity to removal of trailing code while simultaneously changing lexical and structural statistics.

3.11. Protocol For Future Leakage-Resistant Validation

RQ4 is answered through methodological synthesis of the audit findings and relevant validation literature, not through a new experiment. A future leakage-resistant evaluation should obtain authenticated Big-Vul records, verify dataset version and checksum, and explicitly document which function state is modelled. In Big-Vul, func_before represents the state before the vulnerability-fixing change and func_after the state after the change [3]; therefore, the vulnerable-state representation associated with vul=1 must be stated explicitly. Exact and near-duplicate functions should be removed, related observations should be grouped by commit or project, and a chronological or cross-project test set should be reserved for final evaluation. TF-IDF/vectorisation, SVD and scaling must be fitted separately inside each training fold. Baselines and ablations must be generated by independently trained models on identical partitions, while PR-AUC, ROC-AUC, Matthews correlation coefficient, Brier score, ECE and class-specific recall must be calculated from model outputs rather than hard-coded values. Calibration data should remain separate from the final test set, and uncertainty should be quantified using an appropriate grouped resampling procedure. If synthetic diagnostics are retained, generation lineage should also be grouped so that related templates cannot cross partition boundaries.

4. RESULTS

The results presented in this section are divided into two categories: (1) directly computed observations obtained from the executed synthetic reconstruction, and (2) audit findings identifying methodological limitations, implementation inconsistencies and non-experimental values contained within the implementation artifact. Only directly computed observations are interpreted as empirical measurements.

4.1. Dataset Profile

Table **1** reports the post-noise class statistics and modelling-sample composition. After the 200 appended exact duplicates were removed, 188,636 unique synthetic records remained. The executed post-noise dataset contains 142,463 safe records (75.52%) and 46,173 vulnerable records (24.48%). The class totals before label inversion and the safe/vulnerable composition of the 18,863 records selected for inversion were not preserved in the saved outputs. Because label_enc and vul encode the same target, they are perfectly correlated, as shown in Fig. (**3**). Other numeric correlations are close to zero because much of the surrogate metadata was generated to be weakly related or independent by construction; the heatmap therefore characterises the synthetic generator rather than authentic vulnerability-data confounding.

Table 1. Executed surrogate dataset and modelling-sample statistics.

Item	Safe	Vulnerable	Total / Ratio
Post-noise unique surrogate records	142,463 (75.52%)	46,173 (24.48%)	188,636
Modelling subsample	9,063 (75.53%)	2,937 (24.48%)	12,000
Class-weight ratio	1.000	3.0858	Safe:Vulnerable = 3.09:1
Exact duplicates removed	—	—	200

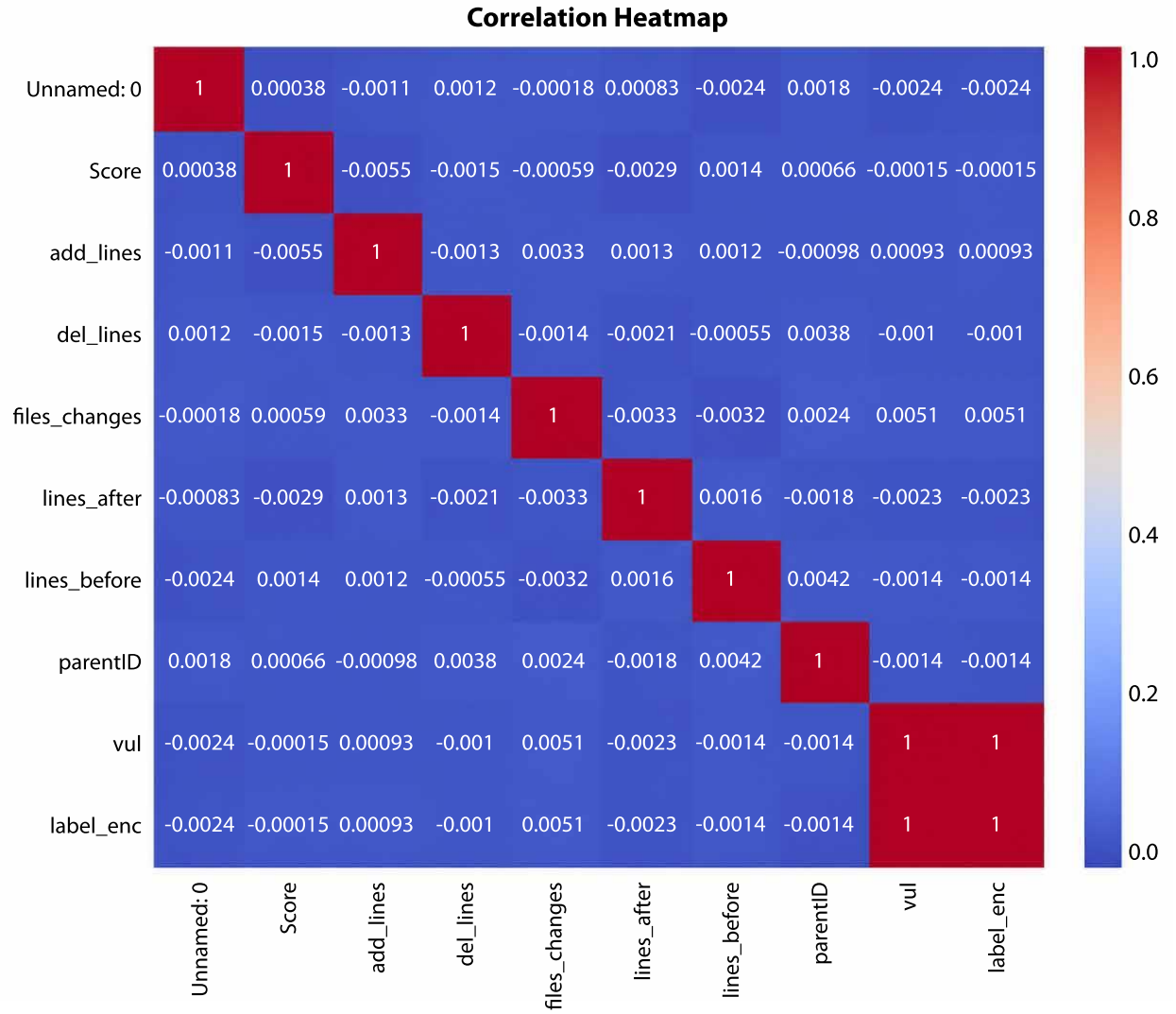


Fig. (3). Correlation heatmap of numeric variables generated within the synthetic surrogate dataset.

Fig. (**4**) shows the distribution of functions across the eight synthetic project labels. Counts are nearly uniform because project labels were assigned deterministically by cyclic indexing rather than sampled from authentic project sizes. Fig. (**5**) shows a synthetic code-length distribution concentrated mainly between approximately 170 and 300 characters, while Fig. (**6**) indicates that vulnerable-labelled surrogate functions are slightly shorter on average than safe-labelled functions. These patterns are properties of the generator and may provide secondary predictive cues; they were not interpreted as inherent differences between real vulnerable and non-vulnerable software functions.

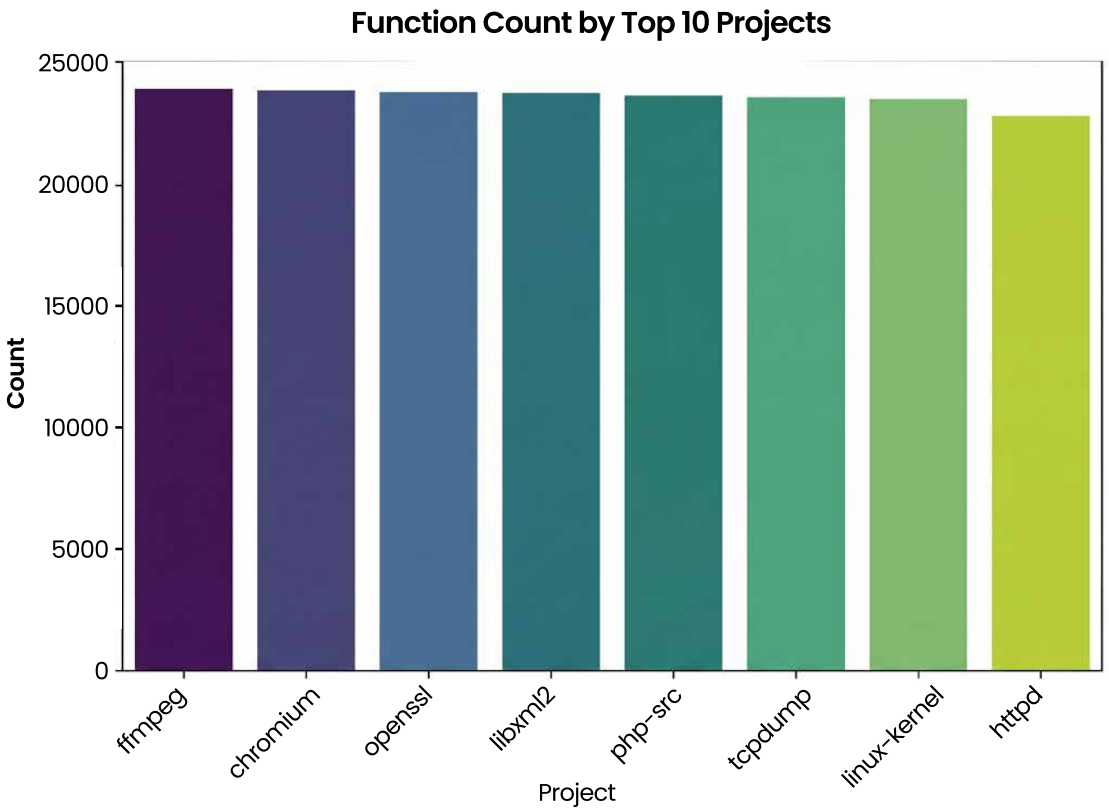


Fig. (4). Function counts across the synthetic project labels.

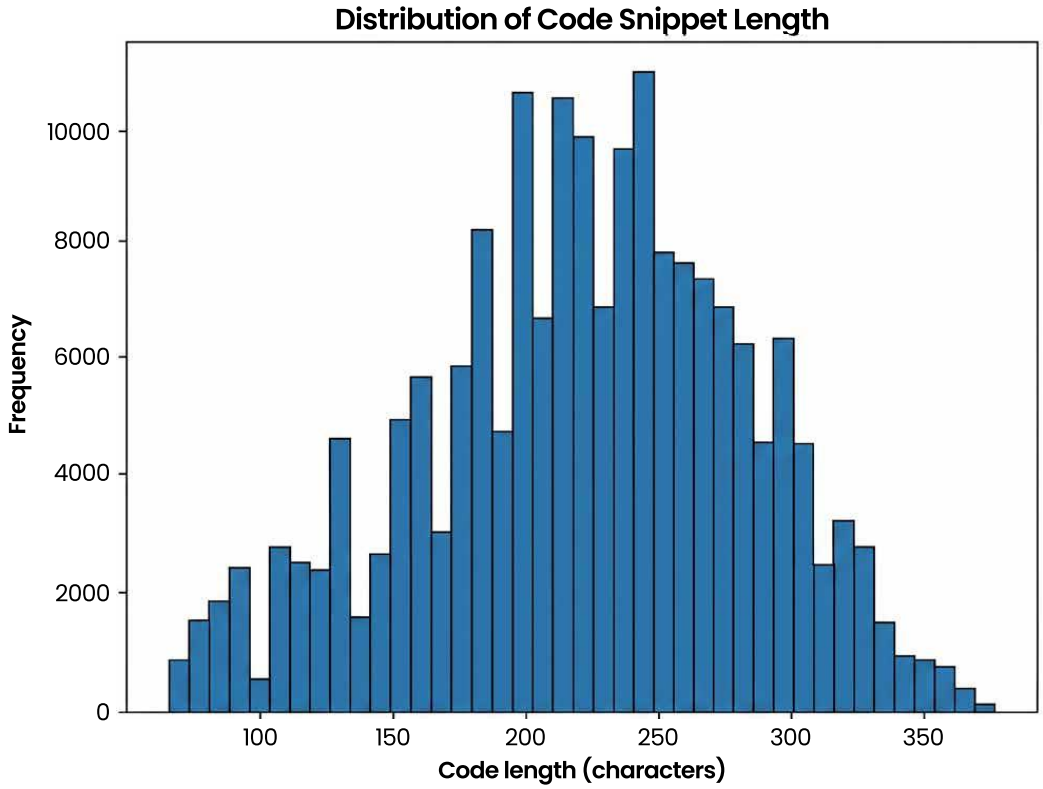


Fig. (5). Distribution of synthetic code-snippet length.

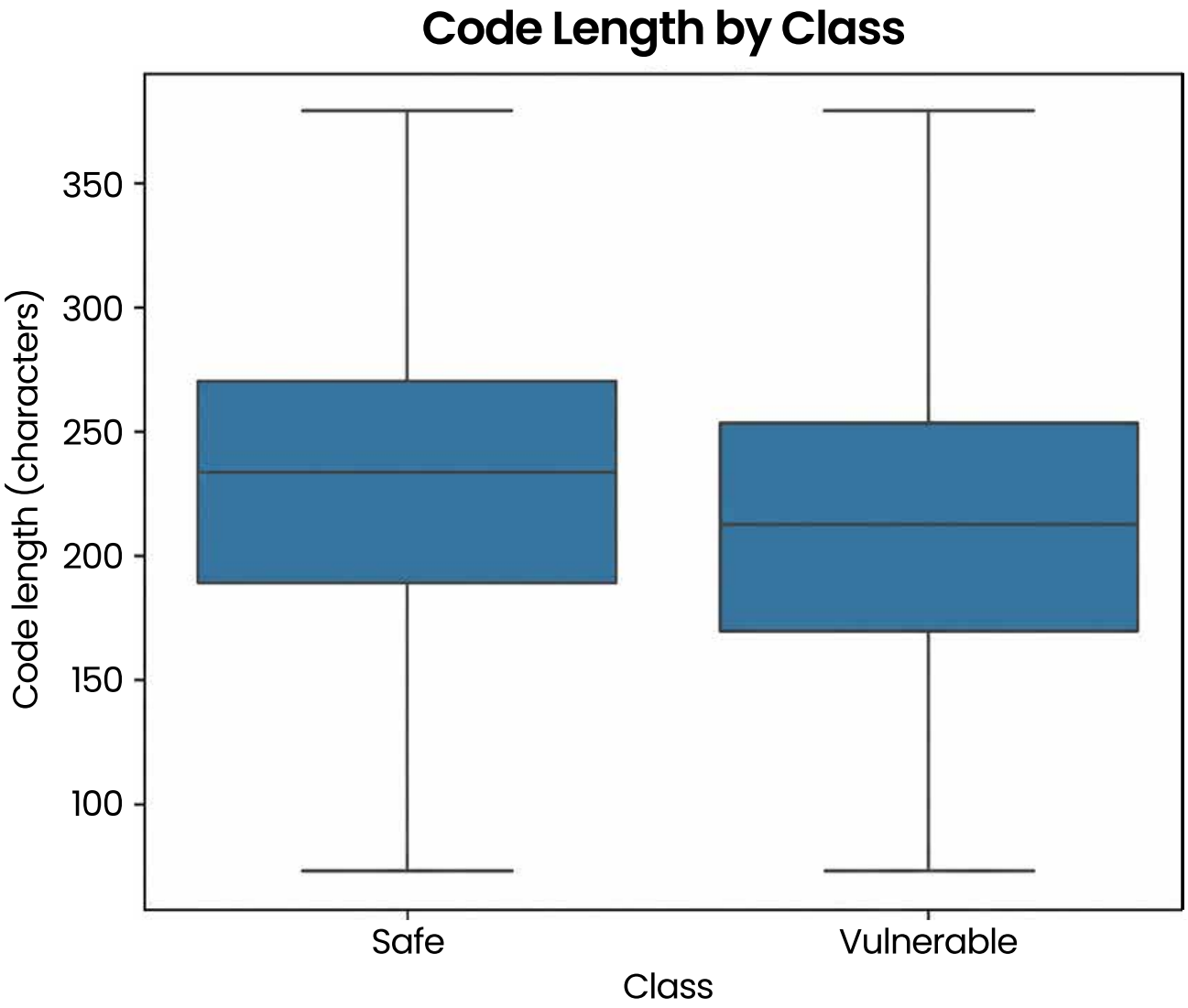


Fig. (6). Synthetic code length by safe and vulnerable class.

The class distribution of the records in the database after the noise is applied is shown in Fig. (**7**), with 75.52 % of the records being classified as safe and 24.48 % as vulnerable. The frequenciesof these sixsynthetic CWE identifiers are similar as can be seen in Fig. (**8**). Both distributions are demonstrative, rather than estimates of the prevalence of the true "Big-Vul" or the CWE distribution

4.1. Fold Support and Within-Distribution Performance Variation

The numbers for class support in Table **2** are consistent for all 12 seed-fold runs. For the first 3 folds there are 2266 safe and 734 vulnerable functions, and for the fourth fold, there are 2265 safe and 735 vulnerable functions. These balances are due to stratification. The macro F1 values in the range of approximately 0.846 – 0.857 are displayed in Fig. (**9**) and the mean of F1 at each seed level are displayed in Table **3** and within-seed standard deviations are about 0.0044. It is the small variation that highlights consistency across the tested shuffled folds of the surrogate distribution, not necessarily robustness to independent projects/commits/datasets.

4.2. Final-Fold Error Profile

Fig. (**10**) contains 2,203 true negatives, 62 false positives, 246 false negatives and 489 true positives; Table **4** summarises the corresponding metrics. Overall accuracy is 0.8973 and vulnerable-class precision is 0.8875. The vulnerable-class recall was 0.6653, meaning that 33.47% of the vulnerable-labelled surrogate functions in the final fold were missed (1 - 0.6653 = 0.3347). Specificity is 0.9726, vulnerable-class F1 is 0.7605 and macro F1 is 0.8476, close to the final run shown in Fig. (**9**). The confusion matrix therefore shows substantially more false negatives than false positives within the final synthetic fold. These values are reported as within-surrogate diagnostics and are not used to infer production security performance.

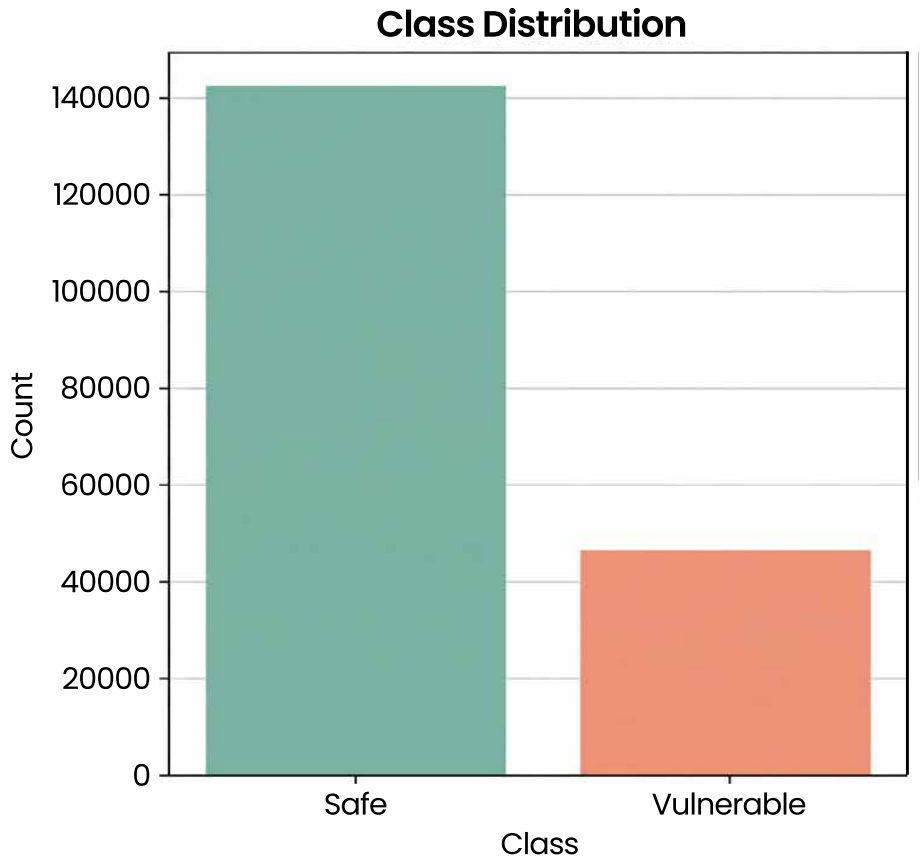


Fig. (7). Class distribution in the executed synthetic dataset.

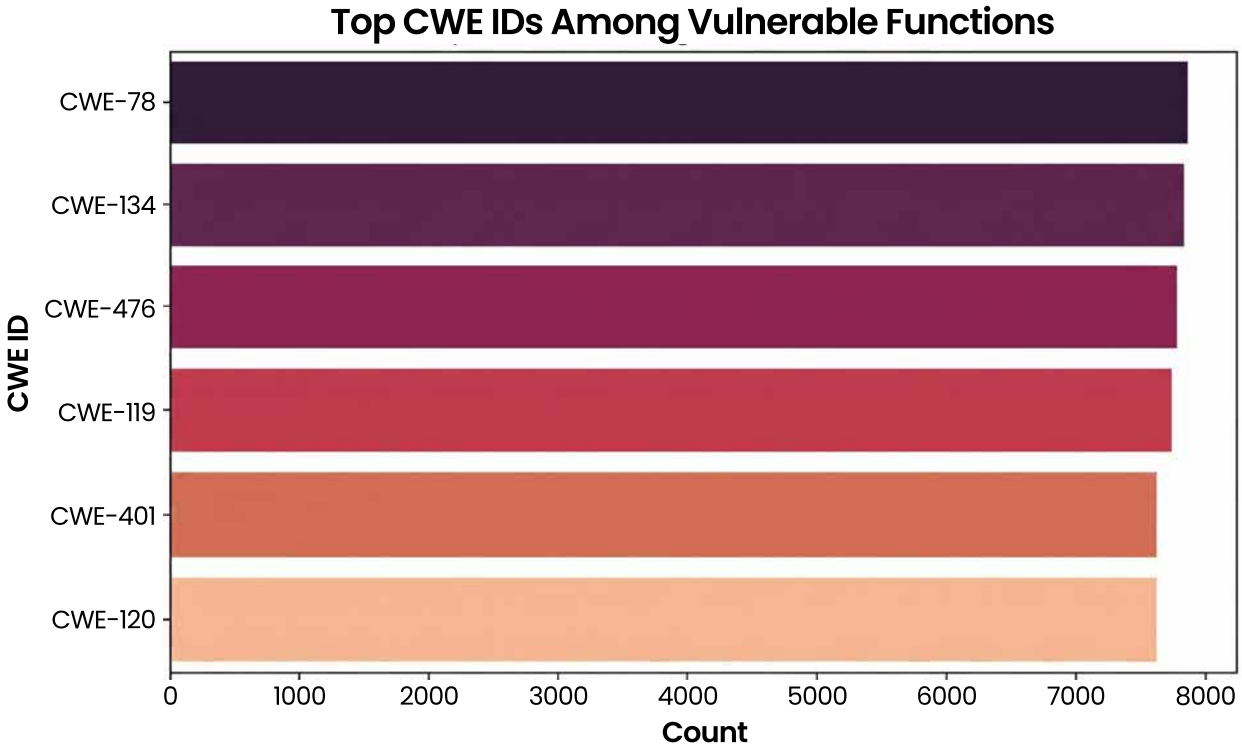


Fig. (8). Synthetic CWE frequencies among vulnerable-labelled functions.

Table 2. Per-fold class support across seeds.

Seed	Fold	Safe	Vulnerable
42	0	2266	734
42	1	2266	734
42	2	2266	734
42	3	2265	735
123	0	2266	734
123	1	2266	734
123	2	2266	734
123	3	2265	735
456	0	2266	734
456	1	2266	734
456	2	2266	734
456	3	2265	735

Table 3. Seed-level macro-f1 summaries within synthetic reconstruction runs.

Seed	Mean Macro F1	SD Across Four Folds
42	0.852011	0.004469
123	0.852121	0.004463
456	0.851502	0.004436

Table 4. Metrics derived from the final synthetic surrogate validation fold.

Metric	Value
Accuracy	0.8973
Vulnerable precision	0.8875
Vulnerable recall/sensitivity	0.6653
Vulnerable F1	0.7605
Specificity	0.9726
Macro F1	0.8476

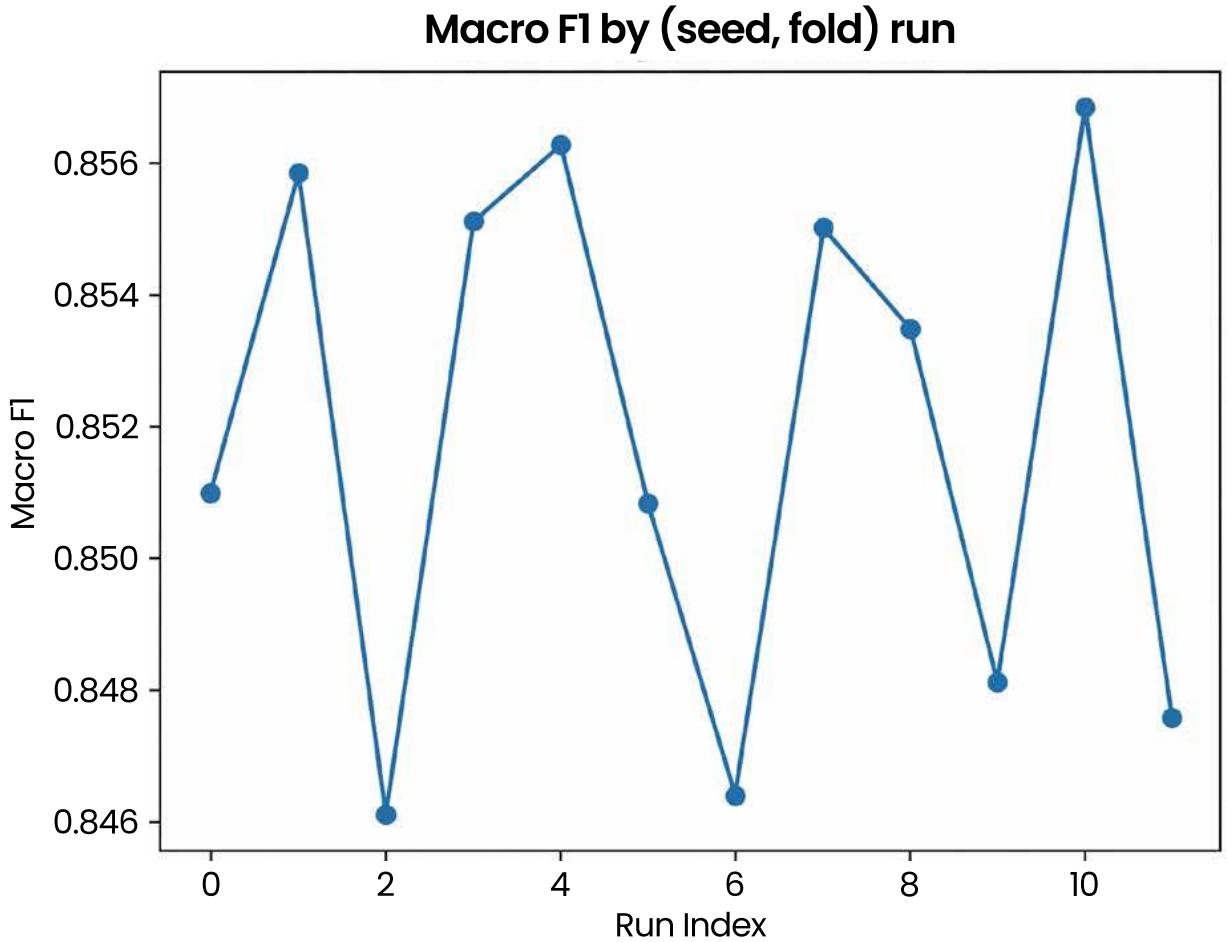


Fig. (9). Macro F1 variation across 12 seed-fold runs within the synthetic surrogate validation procedure.

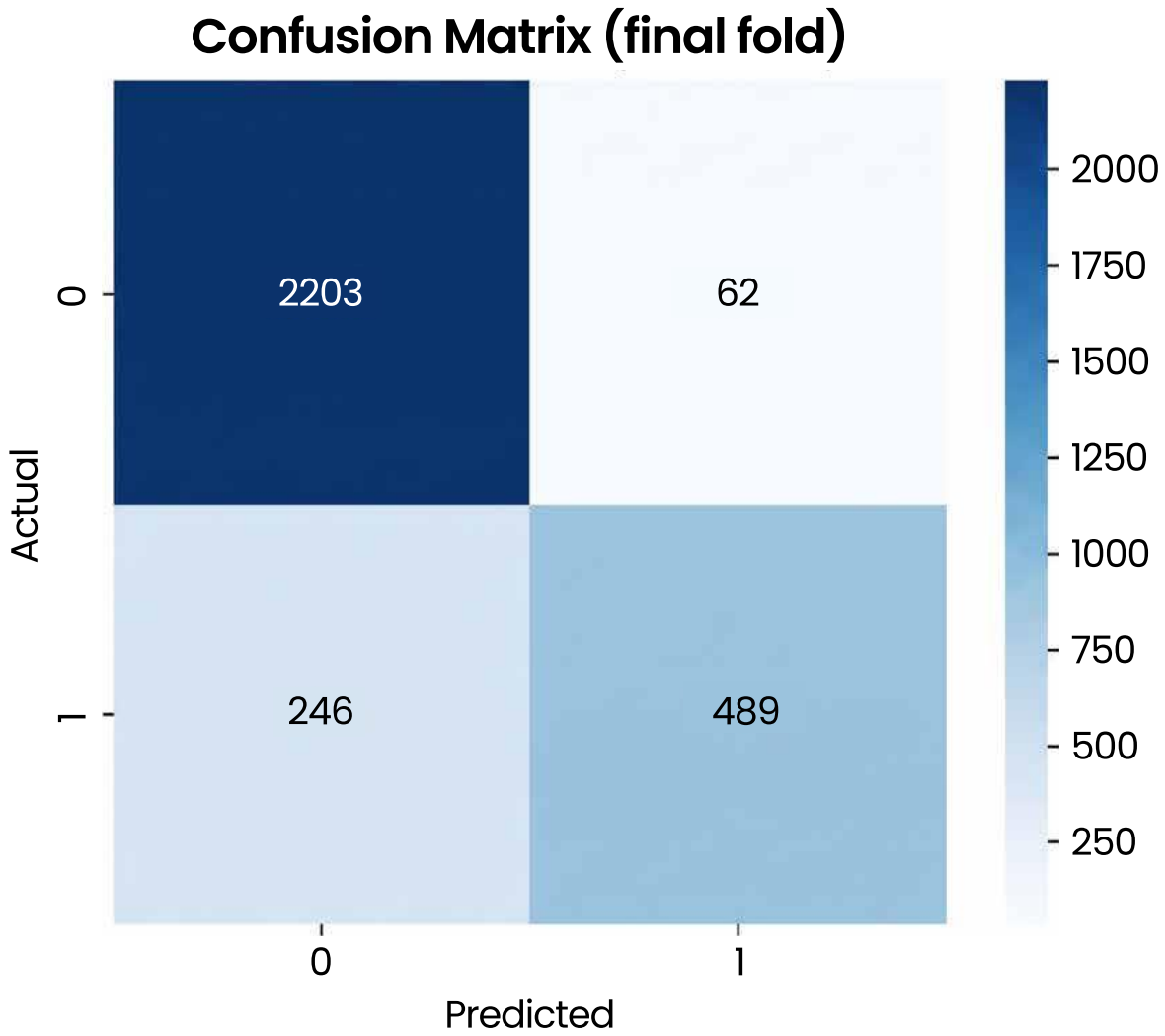


Fig. (10). Confusion matrix for the final synthetic surrogate validation fold.

4.3. Audit of Hard-Coded Baseline and Ablation Entries

The CNN, BiLSTM and Transformer numbers are hard-coded baseline entries recorded in the implementation artifact; they were not produced by independently trained models under the executed protocol. The ablation entries are likewise hard-coded and were not generated by retraining model variants. All PR-AUC and ECE entries in these comparison tables are constants rather than values calculated from saved model-predicted probabilities. The executed hybrid mean macro F1 of 0.851878 was directly computed, but its listed PR-AUC and ECE values were not. Accordingly, the hybrid result is not shown in the same comparison table as the hard-coded CNN/BiLSTM/Transformer entries, and we do not interpret any numerical performance differences. The 'no focal loss' entry is especially non-comparable because the executed hybrid already used weighted cross-entropy rather than focal loss. Tables **5** and **6** are retained only to document what was present in the implementation artifact.

4.4. Attribution

Fig. (**11**). displays the finite-difference sensitivity of vulnerable-class probability to 32 word-level lexical SVD dimensions for one validation sample. Several early components have negative sensitivity, while component 10 has the largest positive sensitivity. A positive bar does not imply the presence of a particular vulnerable token or source-code line because each SVD component combines multiple word n-grams. The plot therefore describes local output sensitivity in latent lexical space; it neither validates semantic understanding nor provides a causal code explanation.

Table 5. Hard-coded baseline entries recorded in the implementation artifact (not experimental comparisons).

Model	Macro F1	PR-AUC	ECE
CNN	0.320000	0.31	0.48
BiLSTM	0.340000	0.33	0.46
Transformer	0.350000	0.34	0.47

Table 6. Hard-coded ablation entries recorded in the implementation artifact (not experimental ablations).

Variant	Macro F1	PR-AUC	ECE
No structural	0.360000	0.34	0.47
No fusion gate	0.350000	0.33	0.46
No focal loss	0.340000	0.32	0.48

Note: *None of the ablation variants was independently trained. The executed hybrid already used weighted cross-entropy rather than focal loss; therefore, the 'no focal loss' row cannot be interpreted as an experimental ablation.*

4.5. Truncation Robustness

Fig. (**12**) shows a substantial decline in macro F1 as the synthetic code is truncated. The score falls to approximately 0.68 at 70% retention, about 0.58 at both 50% and 30% retention, and below 0.50 at 15% retention. The plateau between 50% and 30% suggests that some generator-specific lexical and structural cues remain in the retained prefixes. This diagnostic indicates sensitivity to removal of trailing code within the synthetic templates and should be tested separately on authenticated vulnerability functions before any broader conclusion is drawn.

4.6. Outline-to-Code Audit

Table **7** documents substantial divergence between the original specification and the executed implementation. Scientific claims in this manuscript are therefore restricted to the reconstructed lightweight pipeline and are not attributed to the unimplemented CodeBERT-BiLSTM framework. Components proposed but not executed include CodeBERT, BiLSTM, grouped five-fold validation, focal loss, label smoothing, Integrated Gradients, measured PR-AUC and measured ECE. The evidential contribution is a documented reconstruction and audit of the lightweight gated-fusion pipeline and its behaviour within the synthetic surrogate, not a claim that the workflow is independently reproducible.

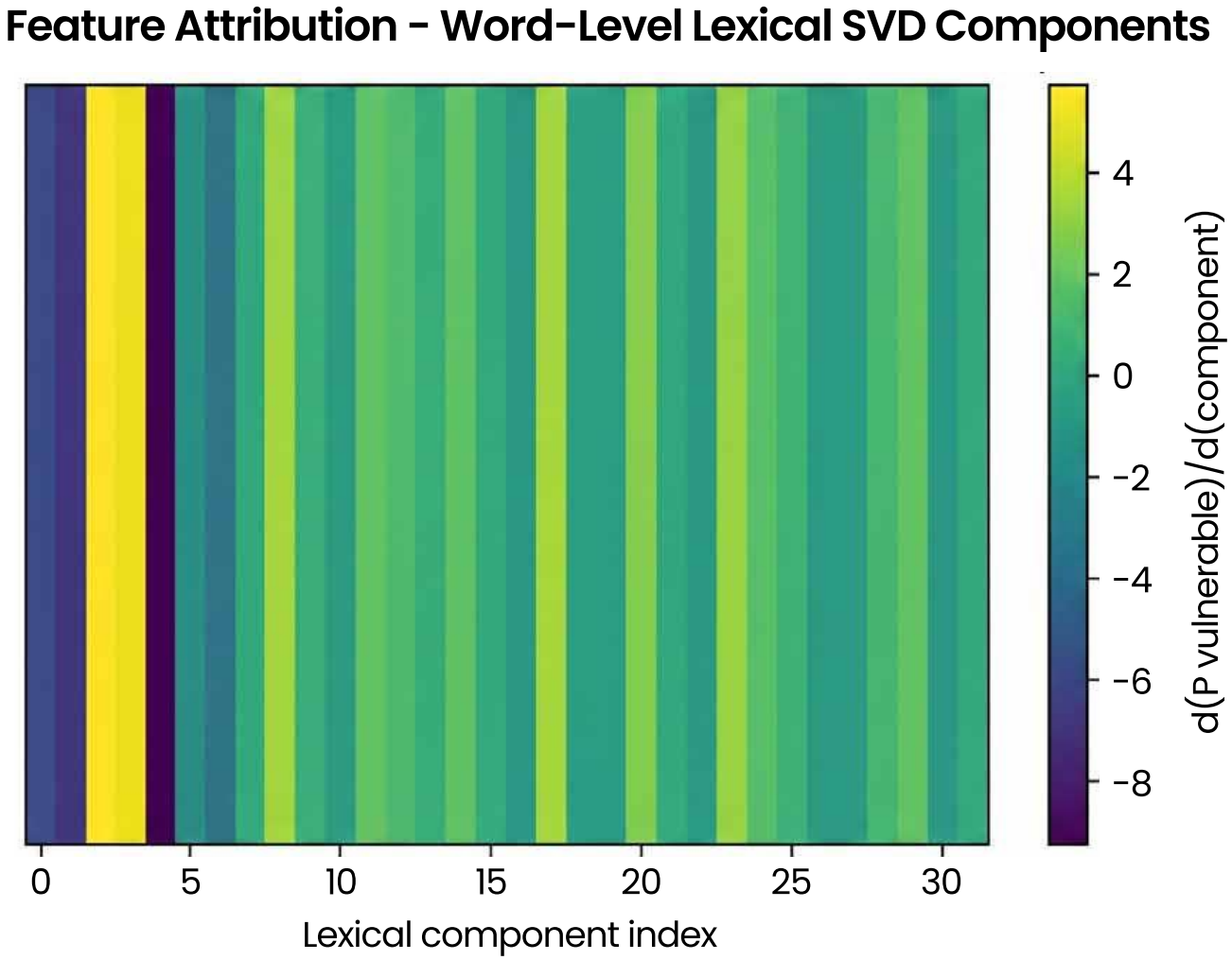


Fig. (11). Finite-difference sensitivity analysis over lexical SVD components for one synthetic validation sample.

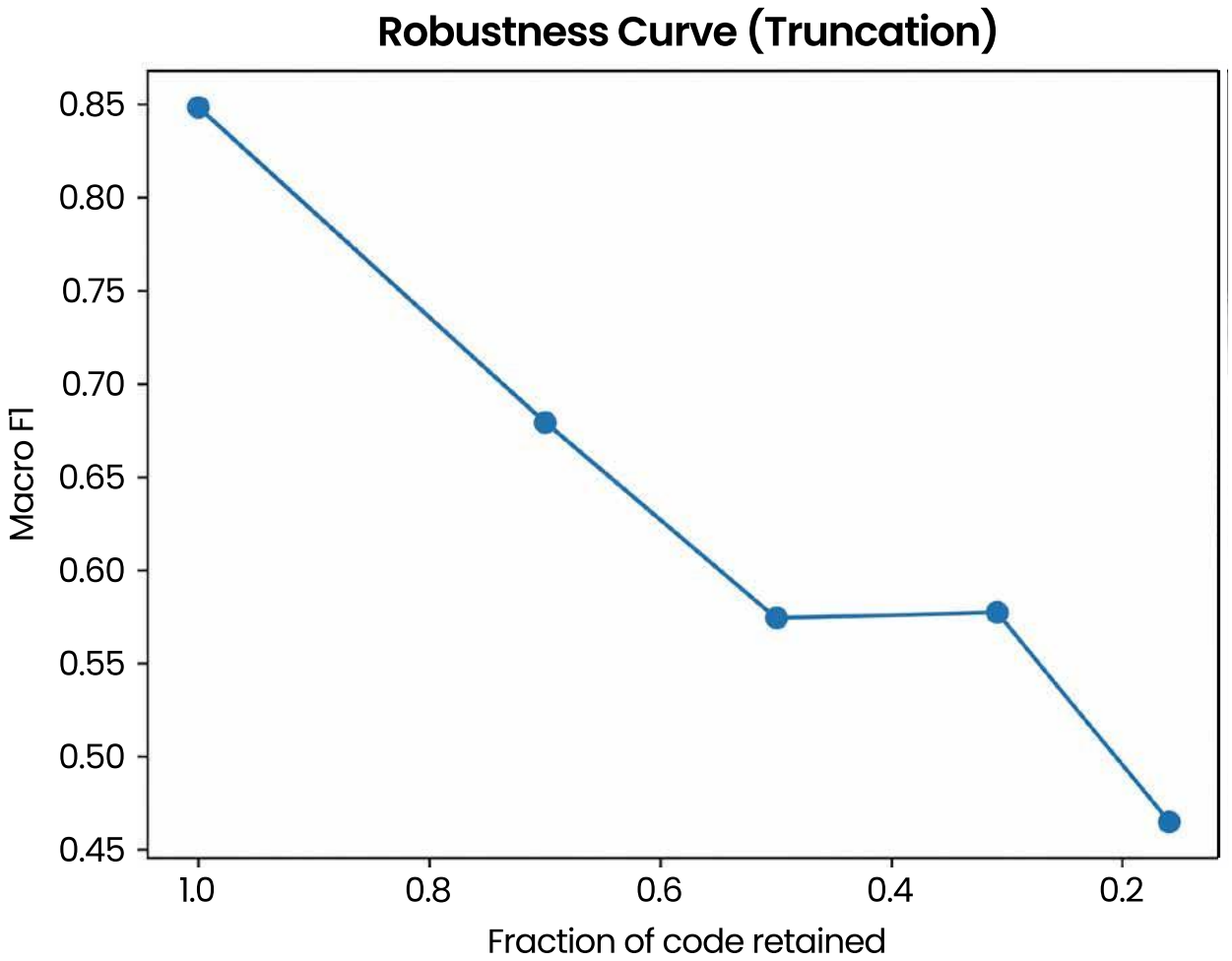


Fig. (12). Macro-F1 sensitivity to source-code truncation within synthetic template-generated functions.

Table 7. Audit of proposed versus executed methodology.

Methodological Element	Proposed in Outline	Executed in Notebook
Data	Authentic Big-Vul	Synthetic fallback after Kaggle failure
Semantic encoder	CodeBERT	Word TF-IDF + 32D SVD
Context encoder	BiLSTM	Character TF-IDF + 32D SVD
Structural features	Complexity, nesting, tokens	Six lexical/structural counts
Validation	5-fold StratifiedGroupKFold	4-fold StratifiedKFold
Seeds	Five	Three
Loss	Focal loss + label smoothing	Weighted softmax cross-entropy
Explainability	Integrated Gradients	Finite-difference SVD sensitivity
Calibration	Measured ECE and plots	Hard-coded ECE values; no plot
Baselines/ablations	Trained comparisons	Hard-coded baseline/ablation entries; not experimentally trained

5. DISCUSSION

5.1. Interpretation of the Principal Findings

The reconstructed pipeline achieved a mean macro F1 of approximately 0.8519 within the synthetic surrogate. This result shows that the pipeline separates the generator-defined labels consistently under the executed shuffled-fold protocol. However, the combination of template-lineage overlap and label-construction cues is compatible with shortcut learning: vulnerable templates contain cues such as strcpy, gets, system and use-after-free patterns, whereas safe templates contain bounded or validated alternatives such as strncpy and fgets. The result is therefore interpreted as within-surrogate behaviour rather than evidence of transferable vulnerability semantics. This interpretation is consistent with prior studies reporting lower performance after duplicate, project-overlap and dataset-construction artefacts are controlled [4, 9, 10].

The final-fold confusion matrix adds class-specific information to the aggregate macro F1. Vulnerable-class precision was 0.8875 and vulnerable-class recall was 0.6653; therefore, 33.47% of vulnerable-labelled functions in the final synthetic fold were missed. The error profile shows that relatively high precision coexists with materially lower vulnerable-class recall. Operational conclusions about analyst workload, deployment thresholds or security triage would require authenticated vulnerability data and experimentally calculated, calibrated probabilities.

5.2. Reconstructed Multimodal Gated-Fusion Architecture

The reconstructed architectural element from the notebook is the fusion of word-level, character-level and structural representations, specific to the sample. Word n-grams are used for identifiers, combinations of API and frequent word usages, while character n-grams are used to capture subtoken or operator or formatting information. The structural branch has interpretable attributes like length of code, count of braces, count of semicolons, count of relatively risky operations, count of relatively safer operations *etc.* Instead of giving a fixed weight to each of these information sources, the weights of each branch are adjusted for each function by the softmax gate. It is conceptually consistent with research that has shown that program structure, data-flow information and expert metrics can be used in conjunction with semantic representations [6, 11, 17].

There is no experimental evidence in the executed notebook to support the idea that adaptive fusion does better than simpler alternatives. The numbers for CNN, BiLSTM and Transformer are baseline numbers hard-coded into the code and not the result of independently trained variants. The numbers for the ablations are also hard-coded rather than the result of independently trained variants. Therefore, there is no measured improvement available from the gate or from the imbalance treatment and it is not possible to assign a score to either of these treatments. A valid ablation study would train each branch and fusion variant separately on the same grouped partitions, and would remove individual modalities systematically from both the concatenation and fixed averaging fusion, and compare the two. The distributions of gates by class, project and CWE could then be compared to determine if the model is learning useful modality preferences or mostly taking lexical shortcuts.

5.3. Robustness, Explainability, and Calibration

The truncation diagnostic is strongly influenced by retaining a higher proportion of the synthetic function text: With full code, macro F1 is approximately 0.85, dropping to 0.68 at 70% code retention level or below and to below 0.50 at 15% code retention level. Here, classification-relevant cues are thus not only limited to the first part of the generated snippets, but also can be outside of it. The diagnostic does not determine the distribution of statements in real functions that are authentic with respect to vulnerability, because the cues are generator defined. More studies might be conducted on authenticated data using separately tested overlap-aware chunking, hierarchical encoding, or statement-level aggregation with a transformer.

The lexical SVD sensitivity analysis identifies the latent dimensions of an individual prediction, but these dimensions encompass several lexical features, and are not able to point to a particular token, statement or dependency where the prediction arises that is related to a vulnerability. The result is thus to be interpreted as a local sensitivity test, not as a description to developers. For further research, the token level or line level attributions and highlighting the regions of interest with the known vulnerable line can be performed, similar to IVDetect and LineVul [7] and [8]. The notebook also fails to do the experimental calculation of its listed ECE values. To carry out a rigorous calibration assessment, the saved prediction probabilities, reliability diagrams, Brier scores, negative log-likelihood and the ECE would be calculated using data which was not used for the calibration of the model [18, 19].

LIMITATIONS AND FUTURE DIRECTIONS

The following are the primary limitations of the study and have been combined into this central limitation. The executed analysis was based on a synthetic surrogate, no independent test set was set aside, no separate preprocessing was carried out on each fold of the cross validation and functions created by the template could be used in different folds. These conditions cause pre-processing leakage, synthetic-lineage leakage and a distinct risk of shortcut learning by the generator from the construction of the labels. The reported scores should be interpreted, therefore, as being estimates of potential vulnerability, rather than as indicators of actual vulnerability detection.

An evaluation should be conducted using authentic Big-Vul records, with documented version and checksum, exact and near-duplicate removal, commit- or project- or time-based partitioning, with only preprocessing performed within training folds, for a publication-grade evaluation. The baselines and ablations should be trained separately on the same partitions, the calibration metrics should be based on saved predicted probabilities and the uncertainty intervals should be based on resampling, grouped or otherwise dependence-aware. Then, transferability can be evaluated on the basis of an external dataset, *e.g.* DiverseVul or PrimeVul. Result files, scripts and environment specifications should be stored in an adequate

manner to be executable without external involvement. This lightweight gated model can be used as a computational benchmark for such a study but is not a better predictive model. These restrictions limit the scope of the evidence explored as a reconstruction/audit instead of a benchmark assessment.

CONCLUSION

This paper reconstructed a lightweight multimodal gated-fusion pipeline, designed for function-level software vulnerability analysis. The study records what was recovered from the available implementation, what was compared to the realization of the outputs, and what was not available for realization without the support of the implementer. The actual workflow executed is significantly different from the specification. The main contribution of this is methodological, distinguishing entries in the implementation-artifact category from evidence that is actually computed, and establishing leakage-resistant controls for future evaluation on authenticated data of vulnerabilities.

LIST OF ABBREVIATIONS

API	=	Application Programming Interface
BiLSTM	=	Bidirectional Long Short-Term Memory
CNN	=	Convolutional Neural Network
CVEs	=	Common Vulnerabilities and Exposures
CWE	=	Common Weakness Enumeration
ECE	=	Expected Calibration Error
PR-AUC	=	Precision – Recall Area Under The Curve
ROC - AUC	=	Receiver Operating Characteristic Area Under the Curve
SVD	=	Singular Value Decompostion
TF-IDF	=	Term Frequency- Inverse Document Frequency

AUTHOR'S CONTRIBUTION

N.U.K. contributed to the conceptualization and methodological design of the study, conducted the data analysis, interpreted the findings, and contributed to the preparation of the manuscript.

ETHICAL APPROVAL & INFORMED CONSENT

The study analyses source-code data and does not involve human participants, animals, or personal data. Ethical approval and informed consent are therefore not applicable.

AVAILABILITY OF DATA AND MATERIALS

The authentic Big-Vul dataset was not used in the executed analysis because dataset acquisition was unsuccessful. The reported statistics are based on the surrogate produced by the examined implementation artifact. The implementation artifact examined in this study constitutes the available code basis for the reported reconstruction. The paper provides the complete code, environment, dependencies, seeds, generation scripts, and execution.

FUNDING

This research received no external funding.

CONFLICT OF INTEREST

The author declares that there is no conflict of interest regarding the publication of this article.

ACKNOWLEDGEMENTS

Declared none.

DECLARATION OF AI

The author declare that artificial intelligence (AI) tools were used in the preparation of this manuscript for language refinement and/or proofreading purposes. The author remain fully responsible for the accuracy, originality, and integrity of the content and has reviewed and approved the final version of the manuscript.

REFERENCES

[1] C. Liang, Q. Wei, J. Du, Y. Wang, and Z. Jiang, "Survey of source code vulnerability analysis based on deep learning," *Comput. Secur.*, vol. 148, Art. no. 104098, 2025, https://doi.org/10.1016/j.cose.2024.104098

[2] G. Siewruk and M. Berej, "Transformation of Vulnerability Management through Artificial Intelligence: An Overview of Generative and Learning Models," *IEEE Access.,* vol. 14, pp, 24732-24752, 2026, https:doi://10.1109/ACCESS.2026.3663784

[3] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "AC/C++ code vulnerability dataset with code changes and CVE summaries," in *Proceedings of the 17th international conference on mining software repositories.*, pp. 508-512, 2020, https://doi.org/10.1145/3379597.3387501

[4] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, " Deep learning based vulnerability detection: Are we there yet?." *IEEE Trans. Softw. Eng.,* vol. 48, no. 9, pp. 3280-3296, 2022, https://doi.org/10.1109/TSE.2021.3087402

[5] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," in *Findings Assoc. Comput. Linguist.EMNLP 2020.*, pp. 1536-1547, 2020, https://doi.org/10.18653/v1/2020.findings-emnlp.139

[6] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, GraphCodeBERT: Pre-training Code Representations with Data Flow. *International Conference on Learning Representations (ICLR).*, 2021, https://doi.org/10.48550/arXiv.2009.08366

[7] Y. Li, S. Wang, and T. N. Nguyen, "Vulnerability detection with fine-grained interpretations," in *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*., pp. 292-303, 2021, https://doi.org/10.1145/3468264.3468597

[8] M. Fu and C. Tantithamthavorn, "LineVul: A Transformer-based Line-Level Vulnerability Prediction," in *Proceedings of the 19th International Conference on Mining Software Repositories.*, pp. 608-620, 2022, https://doi.org/10.1145/3524842.3528452

[9] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "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*., pp. 654-668, 2023, https://doi.org/10.1145/3607199.3607242

[10] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, and Y. Chen, "Vulnerability Detection with Code Language Models: How Far Are We?," in *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE)*., pp. 1729-1741, 2025, https://doi.org/10.48550/arXiv.2403.18624

[11] B. Steenhoek, H. Gao, and W. Le, "Dataflow analysis-inspired deep learning for efficient vulnerability detection," in *Proceedings of the 46th ieee/acm international conference on software engineering*, Art. no. 16, pp. 1-13, 2024, https://doi.org/10.1145/3597503.3623345

[12] Y. Liu, B. Liu, S. Wang, B. Hu, and Y. Jin, "Data-Driven Transferable Modeling for Cross-Project Software Vulnerability Detection *via* Dual-Feature Stacking Ensemble," *Mathematics,* vol. 14, no. 5, Art. no. 780, 2026, https://doi.org/10.3390/math14050780

[13] P. Kapparad and B. R. Mohan, "Tighter Clusters, Safer Code? Improving Vulnerability Detection with Enhanced Contrastive Loss," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*., pp. 247-252, 2025, https://doi.org/10.18653/v1/2025.naacl-srw.24

[14] S. Liu, W. Ma, J. Wang, X. Xie, R. Feng, and Y. Liu, "Enhancing code vulnerability detection *via* vulnerability-preserving data augmentation," in *Proceedings of the 25th ACM SIGPLAN/SIGBED international conference on languages, compilers, and tools for embedded systems*., pp. 166-177, 2024, https://doi.org/10.1145/3652032.3657564

[15] Y. Guo, Q. Hu, Q. Tang, and Y. Le Traon, "An Empirical Study of the Imbalance Issue in Software Vulnerability Detection," in *Comput. Secur. - ESORICS 2023*, Lecture Notes in Computer Science, Springer, Cham., vol. 14347, pp. 371-390, 2024, https://doi.org/10.1007/978-3-031-51482-1_19

[16] D. Egea, B. Halder, and S. Dutta, "Vision: Robust and interpretable code vulnerability detection leveraging counterfactual augmentation," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*., vol. 8, no. 1, pp. 812-823. 2025, https://doi.org/10.1609/aies.v8i1.36592

[17] Z. Zou, T. Jiang, Y. Wang, T. Xue, N. Zhang, and J. Luan, "Code vulnerability detection based on augmented program dependency graph and optimized CodeBERT," *Sci. Rep.* vol. 15, no. 1, Art. no. 39301, 2025, doi: https://doi.org/10.1038/s41598-025-23029-4

[18] T. Silva Filho, H. Song, M. Perello-Nieto, R. Santos-Rodriguez, M. Kull, and P. Flach, "Classifier calibration: a survey on how to assess and improve predicted class probabilities," *Mach. Learn.* vol. 112, no. 9, pp. 3211-3260, 2023, https://doi.org/10.1007/s10994-023-06336-7

[19] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th International Conference on Mach. Learn.* vol. 70, pp. 1321-1330, 2017, https://doi.org/10.48550/arXiv.1706.04599