Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Hybrid Feature Fusion Framework for Just-in-Time Software Defect Prediction using LLM Reasoning and CodeBERT Embeddings

K. Panimozhi

panimozhi.cse@bmsce.ac.in

B.M.S. College of Engineering

Pratham K P

B.M.S. College of Engineering

Pratheeksha Pai

B.M.S. College of Engineering

Sachin Kumar E

B.M.S. College of Engineering

Research Article

Keywords: Just-in-time defect prediction, CodeBERT, large language models, hybrid feature fusion, commit-level analysis, machine learning

Posted Date: November 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11300160/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A Hybrid Feature Fusion Framework for Just-in-Time Software Defect Prediction using LLM Reasoning and CodeBERT Embeddings

K. Panimozhi[1*], Pratham K P[1], Pratheeksha Pai[1], Sachin Kumar E[1]

[1*]Department of Computer Science and Engineering, B.M.S. College of Engineering, Bengaluru, 560019, Karnataka, India.

*Corresponding author(s). E-mail(s): panimozhi.cse@bmsce.ac.in;
Contributing authors: prathamkp.cs23@bmsce.ac.in;
pratheekshapai.cs23@bmsce.ac.in; sachinkumar.cs24@bmsce.ac.in;

Abstract

Just-in-Time (JIT) software defect prediction tries to flag risky commits the moment they are committed, before they can cause problems in a codebase. Most existing approaches rely on simple change metrics, such as how many lines were added or how experienced the developer is, but these numbers say very little about what a change actually does. This paper presents a hybrid framework that looks at a commit from three angles at once: traditional change metrics, a semantic understanding of the code from CodeBERT, and structured reasoning about intent and risk generated by a large language model. We built a multi-language dataset of nearly 60,000 commits from 19 open-source Java, Python, and C/C++ projects, and tested several ways of combining these three sources of information using standard machine learning classifiers. Combining traditional metrics with CodeBERT's code understanding consistently gave the strongest results, clearly outperforming any single source used alone. Adding the LLM-based reasoning on top of that combination did not meaningfully improve things further, though it was genuinely useful when paired with either of the other two sources on its own. Overall, our results suggest that combining structural and semantic information is a practical way to catch defects early, while a third, coarser reasoning layer adds little once the first two are already working together.

Keywords: Just-in-time defect prediction, CodeBERT, large language models, hybrid feature fusion, commit-level analysis, machine learning

1 Introduction

Just-in-Time (JIT) defect prediction [1] evaluates each commit individually at the moment it is pushed, rather than waiting for file or module-level aggregation, so that a developer or a CI/CD gate can act on a quality warning before the change is merged. Formally, given a repository history of commits

$$C = \{c_1, \ldots, c_N\},$$

a JIT-SDP model learns a classifier

$$f : C \to \{0, 1\}$$

where

$$f(c_i) = 1$$

denotes a bug-inducing commit, using a training set

$$D = \{(x_i, y_i)\}_{i=1}^{N_{\text{train}}}$$

in which y_i is produced by the SZZ bug-inducing-commit tracing algorithm [3].

Two limitations recur across the JIT-SDP literature. First, the widely used Kamei feature set [1, 2] - lines added/deleted, files touched, developer experience, change entropy, and related counts - treats a commit as an uninterpreted numeric summary. Two, commits of identical size can have identical metric vectors and yet only one introduces a defect, because the metrics say nothing about *what* changed. Second, even the transformer-based embeddings that address this gap, such as CodeBERT [4], encode the syntax and surface semantics of a diff but not the developer's *intent* or an explicit estimate of *risk*. Large language models can supply exactly this kind of structured, human-interpretable reasoning from a commit message and diff [5], but prior work has used LLMs either as standalone commit classifiers or as relabelling tools [6], not as a feature stream fused with structural and embedding-based representations inside one classifier.

This paper is organised in two parts that build directly on one another. The first part (Section 2) is a short systematic literature review: we screened over 60 candidate studies drawn from IEEE Xplore, ACM Digital Library, Scopus, and arXiv, together with an internally maintained review spreadsheet used to triage candidates by problem statement, methodology, and reported results, and retained 37 primary studies published between 2018 and 2026 that met our inclusion criteria (primary empirical studies addressing commit-level defect prediction with a reported quantitative evaluation). From this review we distil three recurring gaps - semantic shallowness of structural features, the absence of reasoning-aware features, and near-exclusive single-language evaluation - summarised in a systematic gap-analysis table (Table 4). The second part (Sections 3–6) reports the design, implementation, and measured evaluation of a hybrid framework built to close exactly these three gaps, culminating in a full trimodal fusion evaluation that directly tests whether all three feature streams are worth combining at once.

Concretely, our contributions are:

1. A short systematic review of 37 primary JIT-SDP studies (2018-2026), synthesised into a comparative table of representative approaches (Table 1) and a three-gap analysis (Table 4) that directly motivates the proposed architecture.
2. A 59,996-commit, three-language (Java, Python, C/C++) benchmark mined directly from 19 open-source repositories, labelled with the SZZ (Śliwerski, Zimmermann, and Zeller) algorithm, and split project-wise and chronologically (70/15/15, i.e. 41,998/8,998/9,000 commits) so that no test-time commit ever informs training.
3. A reproducible feature-extraction pipeline for (a) 12 structural JIT metrics, (b) 384-dimensional CodeBERT embeddings obtained via PCA fitted only on the training split (99.43% variance retained), and (c) a seven-field structured LLM reasoning representation extracted with a quantized Qwen3 model and cached per commit.
4. A complete empirical evaluation of all seven feature configurations implied by the three modalities - JIT-only, CodeBERT-only, LLM-only, JIT+CodeBERT, JIT+LLM, CodeBERT+LLM, and full JIT+CodeBERT+LLM trimodal fusion - with Random Forest, XGBoost, and (for the blended and trimodal configurations) LightGBM, evaluated with accuracy, precision, recall, F1, MCC, ROC-AUC, and PR-AUC on a strictly held-out test set, including pairwise comparisons that isolate each stream's marginal contribution.
5. A documented account of the engineering constraints of running LLM feature extraction under a shared, fractional-GPU (NVIDIA MIG) environment, including the model and configuration-selection decisions that made a 59,996-commit extraction run practical, offered here as a reproducibility record for similarly resource-constrained groups.

The remainder of the paper is organised as follows. Section 2 presents the systematic review and gap analysis. Section 3 describes the proposed architecture, including the high-level design and detailed system architecture (Figs. 1 and 2). Section 4 documents dataset construction. Section 5 describes the experimental setup. Section 6 reports results for Experiments 1-7, including per-experiment methodology and pairwise stream comparisons. Section 7 discusses findings, engineering lessons, and limitations. Section 8 concludes and outlines further planned ablations.

2 Related Work

2.1 Review protocol

We searched IEEE Xplore, ACM Digital Library, Scopus, and arXiv with query *("just-in-time" OR "JIT") AND ("defect prediction" OR "bug prediction") AND ("deep learning" OR "CodeBERT" OR "language model" OR "feature fusion")* restricting to studies published in English between 2018 and 2026. Candidates were first triaged into a shared review spreadsheet recording, by paper, the problem statement addressed, methodology, quantitative results, and proposed future work, to ensure that overlapping or clearly out-of-scope studies would be filtered before full-text screening. Of those candidates carried through to full-text review, we retained 37 primary empirical studies on commit-level defect prediction with a reported quantitative evaluation, spanning five informal categories: traditional machine learning, deep

learning, transformer/LLM-based methods, cross-project and online learning, and specialised/explainability-focused studies. We grouped our review around four questions: (RQ1) what feature-engineering strategies are used in state-of-the-art JIT-SDP; (RQ2) how have deep learning and pre-trained transformers improved predictive performance; (RQ3) what role can LLM reasoning play in extracting contextual features; and (RQ4) what gaps remain, and how can hybrid feature fusion address them.

2.2 Metric-based and deep-learning approaches

The earliest and still most widely used feature set is the 14-attribute Kamei set [1], covering change size (lines added/deleted), diffusion (files/directories/subsystems touched), purpose, developer history, experience, change entropy, and inter-change age. Kamei et al. [2] later showed that periodic developer-activity signals carry information beyond static per-commit counts, and Haldar and Capretz [7] confirmed through feature-importance analysis that developer experience and change entropy are consistently strong predictors across project contexts. A cluster of ensemble variants strengthens this line of work without changing its representational limits: hybrid feature-selection with a stacked ensemble [8], nested stacking with heterogeneous feature selectors [9], ensemble learning combined with near-miss under-sampling [10], a five-phase pipeline with a purpose-built Chunk Balancing Algorithm for imbalance [23], and a CatBoost model trained on 2.4 million commits across 33 heterogeneous repositories that isolates file age, revision count, and entropy as the strongest predictors [24]. Method-level replication work is a useful caution here: Chowdhury et al.'s evaluations of method-level bug prediction [25] show that headline accuracies reported under random splits drop substantially once concept drift and time-sensitive evaluation are enforced - one of the reasons the present work insists on a strictly chronological split (Section 5).

Deep learning moved feature engineering from hand-crafted counts to representations learned directly from commit tokens. DeepJIT [11] was the first end-to-end CNN to jointly encode commit messages and diffs, improving AUC by roughly ten points over metric baselines. SimCom++ [12] showed that early or late fusion of expert-designed features with a learned deep representation improves F1 by up to 17.9% over single-modality baselines - an early, encouraging data point for exactly the fusion hypothesis this paper tests directly in Section 6. Related JIT-adjacent work broadens the picture beyond bug-count prediction: a commit-level study of architectural change found that roughly 27% of commits in a 77,000-commit, 30-project Java corpus carried architectural implications [26], reinforcing that a commit's structural footprint and its downstream consequences are not the same thing, and a manual GitHub-issue study of "human-centric" defects found that a meaningful share of reported defects trace to usability and accessibility concerns that standard change-metric features cannot see at all [27].

Table 1 places five representative approaches - spanning metric-based, deep-learning, transformer-embedding, LLM-based, and bimodal-fusion designs - alongside the framework proposed in this paper, to make the representational gap each successive approach closes, and the gap the proposed trimodal design targets, explicit in one place.

Table 1: Comparison of representative JIT-SDP approaches by primary feature representation, learning model, and key limitation.

Approach	Primary features	Learning model	Key limitation
Kamei et al. [1]	Handcrafted JIT metrics (change churn, developer activity)	RF, LR	Relies only on structural metrics; cannot capture semantic meaning or developer intent.
DeepJIT [11]	Commit messages and diffs as raw token sequences	CNN	Captures local textual patterns but lacks global semantic understanding and reasoning.
Yadav et al. [13]	CodeBERT embeddings + software metrics	Stacking ensemble	Uses transformer embeddings but no LLM-based reasoning or structured intent features.
Kim et al. [5]	LLM-derived semantic features	LLM as classifier	Reasons about intent in isolation; does not integrate structural or embedding features.
SimCom++ [12]	Expert features + deep semantic features	Fusion-based DL	Bimodal fusion only; no LLM-driven reasoning or explainable risk signal.
This work	JIT metrics + CodeBERT embeddings + LLM reasoning features	RF / XGBoost / LightGBM	Empirically evaluates structural, semantic, and reasoning-level representation jointly via trimodal fusion.

2.3 CodeBERT and transformer embeddings

CodeBERT [4] is pre-trained on paired code and natural-language text using masked-language modelling and replaced-token detection, and produces a 768-dimensional `[CLS]` embedding per input. Because 768 dimensions is large relative to the number of available training commits, PCA-based dimensionality reduction is common [15]. Yadav et al. [13] combined CodeBERT embeddings with a 22-classifier stacking ensemble to reach 81.08% median accuracy across 41 Apache repositories, and Abu Talib et al. [14] showed that parameter-efficient LoRA fine-tuning of CodeBERT/CodeT5 is sufficient to match full fine-tuning at a fraction of the compute cost-directly relevant to our own GPU-constrained deployment, discussed in Section 7. JIT-Align [15] addressed long-diff truncation by ranking file diffs with FAISS similarity before encoding, and Pujar et al. [28] analysed the 1.3-million-sample D2A vulnerability dataset with ML ensembles combined with C-BERT representations, confirming that transformer-based representations outperform raw token features even outside the JIT-SDP setting proper. At the explainability end, Aleithan [29] showed that standard model-agnostic explanation methods such as LIME fail to produce meaningful explanations for commit-level models despite succeeding at file level - a caution we return to when discussing the interpretability of the LLM reasoning stream (Section 7).

Table 2 summarises the CodeBERT and transformer-embedding studies discussed above, alongside their reported datasets and headline results, to make the progression from raw embeddings to efficient fine-tuning to long-diff handling to explainability limits easy to compare at a glance.

Table 2: Summary of the CodeBERT and transformer-embedding literature reviewed in Section 2 (RQ2).

Study	Year	Approach	Dataset	Key reported result
Feng et al. (CodeBERT) [4]	2020	Bimodal pre-training via masked-LM + replaced-token detection	CodeSearchNet	Establishes the 768-D `[CLS]` code/NL embedding used throughout this line of work
Yadav et al. [13]	2025	CodeBERT (code + comments) + 22-classifier stacking ensemble	41 Apache repositories	81.08% median accuracy
Abu Talib et al. [14]	2024	Parameter-efficient LoRA fine-tuning of CodeBERT/CodeT5	ApacheJIT	Matches full fine-tuning cost-effectively; +10–12% F1 over BiLSTM
Chen et al. (JIT-Align) [15]	2025	CodeT5 + FAISS-ranked diff selection to fix truncation	Multiple OSS projects	+3.1–9.6% F1 from resolving long-diff truncation
Pujar et al. [28]	2024	ML ensembles + C-BERT representations	D2A (1.3M samples)	Transformer representations outperform raw token features
Aleithan [29]	2021	LIME / iBreakDown post-hoc explanation	Qt, OpenStack	Explanation methods that work file-level fail at commit level

2.4 LLM-based reasoning features

Kim et al. [5] showed that LLM-based commit classification reduced bug-introduction rates by 36% in an industrial deployment, and Semantic SZZ [6] used GPT to re-label ambiguous bug-fix commits, reducing mislabelled defect-inducing commits by up to 21% on PostgreSQL. Harzevili et al.'s broader review of ML-based vulnerability detection [30] similarly finds LLM-based methods increasingly applied wherever reasoning about code intent, rather than pattern matching alone, is required. These results establish that LLMs can extract useful contextual signal from commit text, but in every case the LLM acts as a standalone classifier or a relabelling tool rather than as one of several feature streams feeding a downstream classifier - the specific gap this paper's Experiments 3, 5, 6, and 7 are designed to probe.

Table 3 summarises the three LLM-based studies discussed above. Note how thin this table is relative to Table 2: this asymmetry is itself evidence for Gap G2 in Table 4 - LLM reasoning is a comparatively young direction in JIT-SDP, and none of the three uses the LLM as a fused feature stream alongside structural or embedding-based representations, which is exactly the role Stage 2c of the proposed framework (Section 3) gives it.

2.5 Cross-project, online, and specialised methods

A separate strand of the literature addresses generalisation rather than representation. TSBoostDF [31] applies TrAdaBoost transfer learning for cross-project prediction; Feyzi and Daneshdoost study deep active learning [32], and Cabral et al. study periodic

Table 3: Summary of the LLM-based reasoning literature reviewed in Section 2 (RQ3).

Study	Year	Approach	Dataset/context	Key reported result
Kim et al. [5]	2025	LLM-based commit-intent classification (industrial deployment)	Ubisoft CleverCommit, GitHub history	36% reduction in bug-introduction rate
Peng et al. (Semantic SZZ) [6]	2025	GPT-based semantic re-labelling of ambiguous bug-fix commits	PostgreSQL	Up to 21% fewer mislabelled bug-inducing commits
Harzevili et al. [30]	2024	Systematic review of ML/LLM-based vulnerability detection	Multiple CVE datasets	LLM-based reasoning increasingly outperforms classical ML on intent-heavy tasks

offline retraining versus continuously updating online learners [33]; JNAI [34] adapts JIT-SDP to non-stationary, imbalanced data streams under concept drift; federated learning has been used for privacy-preserving cross-repository training [35]; and Li et al. [36] show empirically that no single class-imbalance sampler is universally best, with the optimal choice depending on whether the target metric is classification or ranking-oriented. ApacheJIT [17] remains the standard large-scale, SZZ-labelled Java benchmark that informed the mining strategy of the Java portion of the dataset built for this paper (Section 4).

2.6 Research gap analysis

Synthesising the reviewed studies against RQ4 surfaces three persistent, mutually reinforcing gaps, summarised in Table 4.

These three gaps motivate the hybrid trimodal fusion architecture described next: rather than treating structural metrics, semantic embeddings, and LLM reasoning as competing representations, the proposed framework treats them as complementary streams to be fused, and evaluates every pairing and the full combination on a deliberately multi-language, strictly chronological benchmark.

3 Proposed Framework

3.1 High-level design

Figure 1 shows the high-level design of the framework at the level a project stakeholder would read it: raw commits enter from GitHub repositories, are routed through three parallel, independent feature-extraction paths (JIT metrics, CodeBERT embeddings, LLM reasoning), converge at a fusion stage, and are scored by an ensemble classifier to produce a Buggy/Bug-Free decision that a CI/CD pipeline can act on.

3.2 Detailed system architecture

Figure 2 expands the high-level view into the five concrete pipeline stages implemented for this paper - data collection and pre-processing, parallel feature extraction, feature

Table 4: Systematic summary of the three research gaps identified across the reviewed literature and how the proposed framework addresses each.

Gap	Description	Representative evidence	How this work responds
G1: Semantic shallowness	Metric-based models [1, 2, 9, 24] reduce a commit to numeric counts, discarding what actually changed and why; identically sized commits can carry opposite risk.	[1, 24, 25]	Adds a 384-D CodeBERT semantic embedding of the message and diff, fused with (not replacing) the 12 JIT metrics.
G2: Absent reasoning features	Even embedding-based models [13–15] encode structure and surface semantics but not developer intent, estimated risk, or change type; the interpretability gap is confirmed directly [29].	[5, 13, 14, 29]	Adds a seven-field, confidence-scored LLM reasoning representation (intent, risk, complexity, scope, test impact, security risk, change type) as an explicit third stream, and empirically tests it alone, paired with each other stream, and fused with both.
G3: Single-language evaluation	The large majority of JIT-SDP studies evaluate on Java-only corpora (Qt, OpenStack, ApacheJIT), leaving generalisability to multi-language codebases largely untested.	[16, 17]	Builds and evaluates on a 59,996-commit, three-language (Java, Python, C/C++) benchmark spanning 19 repositories (Section 4).

convergence via fusion, model training and predictive inference - and is the diagram we refer back to throughout Sections 3–6 when describing where a given step happens in the pipeline.

The five stages are described next.

3.2.1 Stage 1: Data collection and preprocessing

Commit-level artefacts (a unique `commit_id`, commit message, unified diff, and JIT metrics) are mined from open-source GitHub repositories in three languages. Bug-inducing labels follow the SZZ algorithm [3]: bug-fix commits are identified by keyword matching on commit messages (`fix`, `bug`, `error`, `issue`, `patch`, `resolve`, `hotfix`), and the corresponding bug-inducing commit is traced via `git blame` line annotation. Section 4 details the specific repositories mined, the resulting commit counts, and the labelling procedure.

3.2.2 Stage 2a: JIT metric extraction

Twelve structural and historical change-level metrics are computed per commit, following the Kamei benchmark [1]: lines added (`la`), lines deleted (`ld`), files changed

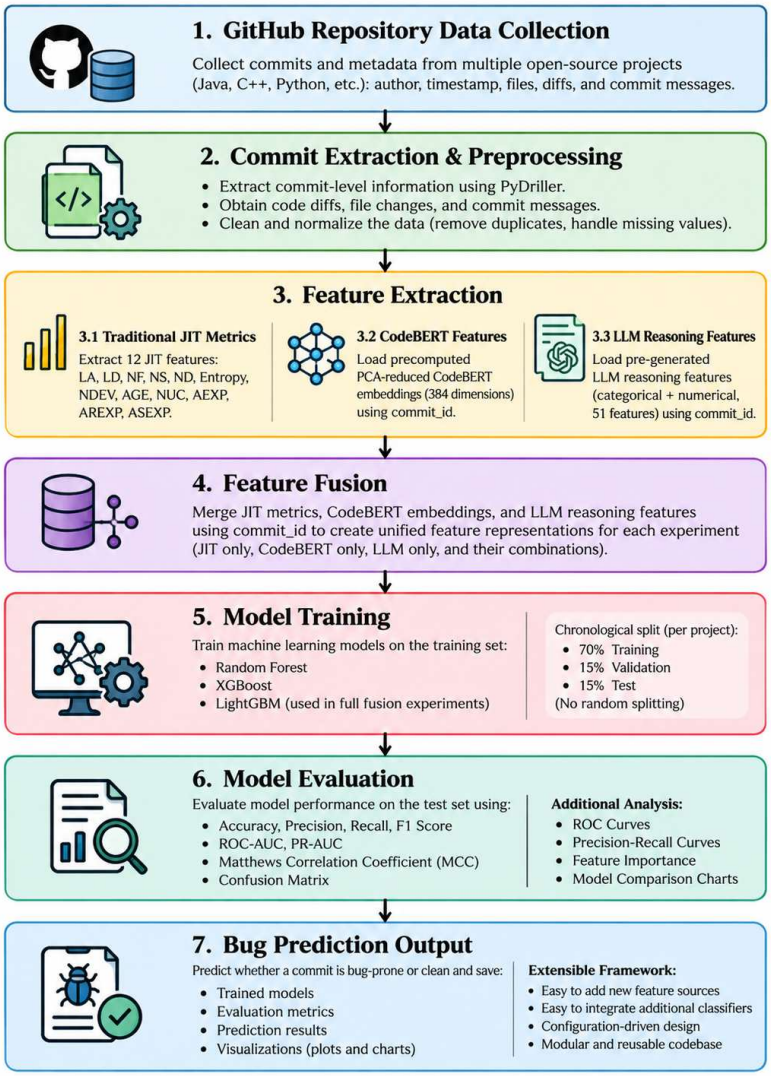


Fig. 1: High-level design of the proposed hybrid feature fusion framework for JIT software defect prediction.

(`nf`), directories changed (`nd`), subsystems changed (`ns`), change entropy (`ent`), number of distinct developers touching the file history (`ndev`), age since last change (`age`), number of unique prior changes to the touched files (`nuc`), and three developer-experience measures - author experience (`aexp`), recent author experience (`arexp`), and author subsystem experience (`asexp`). Missing values are imputed with the training-split median only, and all twelve features are z-score standardised using training-split statistics before fusion.

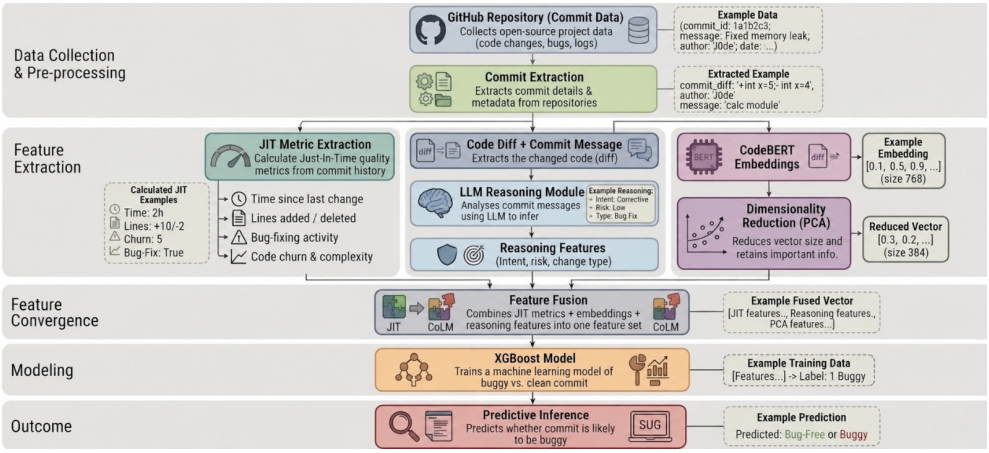


Fig. 2: Detailed system architecture: data collection, parallel JIT/CodeBERT/LLM feature extraction, seven-way fusion sweep, model training, and predictive inference.

3.2.3 Stage 2b: CodeBERT semantic embedding

The concatenated commit message and diff is tokenised and fed to `microsoft/codebert-base` [4], truncated to a fixed token budget, and the final-layer `[CLS]` hidden state is retained as a 768-dimensional embedding. Because 768 dimensions is large relative to the training-split size and introduces collinearity when concatenated with the other streams, PCA reduces the space to 384 components. Critically, the PCA transform is fitted on the training split only (41,998 commits) and then applied unchanged to the validation and test splits, preventing information leakage from future commits into the projection itself. In our implementation, the fitted 384-component projection retained 99.43% of the training-set variance (Table 7).

3.2.4 Stage 2c: LLM reasoning feature extraction

Each commit's message and diff are submitted to the Qwen3-4B large language model, which returns seven categorical fields - `intent`, `change_type` (referred to as `change` in the dataset), `risk_level`, `complexity`, `scope`, `test_impact`, and `security_risk` - together with a `confidence` and `margin` score for each field, for 24 columns per commit including provenance metadata. Unlike free-form prompting, the production pipeline uses candidate-label likelihood scoring, comparing the next-token probabilities of the allowed labels for each field rather than generating free-form text. This produces a compact semantic feature representation without storing chain-of-thought or natural-language reasoning. Section 4 documents the model-selection and inference-configuration process used for this extraction.

3.2.5 Stage 3: Feature fusion

The three streams are combined per commit as

$$\mathbf{f} = [\mathbf{f}_{\text{JIT}} \,\|\, \mathbf{f}_{\text{BERT}} \,\|\, \mathbf{f}_{\text{LLM}}]\,, \tag{1}$$

where $\mathbf{f}_{\text{JIT}} \in \mathbb{R}^{12}$ are the standardised JIT metrics, $\mathbf{f}_{\text{BERT}} \in \mathbb{R}^{384}$ are the PCA-reduced CodeBERT embeddings, and $\mathbf{f}_{\text{LLM}} \in \mathbb{R}^{51}$ is the one-hot-encoded categorical representation of the seven reasoning fields together with their fourteen confidence/margin scores. This paper reports results for every non-trivial subset of Eq. 1 - each stream alone (Experiments 1–3), every pair (Experiments 4–6), and the full concatenation (Experiment 7) - giving a complete picture of each stream's individual and joint contribution (Section 6).

3.2.6 Stage 4: Model training

Random Forest and XGBoost [18] classifiers are trained on each fused representation using the class-imbalance handling and hyperparameters described in Table 7. In addition, LightGBM [19] is trained alongside them for the three-model blends (4F and 7F) and for Experiments 7-9, where combining different classifiers and different feature streams is most likely to pay off.

3.2.7 Stage 5: Predictive inference

At inference time a new commit is scored by extracting $\mathbf{f}_{\text{JIT}}$ from repository metadata, computing $\mathbf{f}_{\text{BERT}}$ through the cached PCA-transformed CodeBERT encoder, and retrieving $\mathbf{f}_{\text{LLM}}$ from the cached (or freshly computed) LLM reasoning pass. The fused vector is passed to the trained classifier, which returns a defect-probability score and a binary Buggy/Bug-Free label.

4 Dataset Construction

4.1 Repository selection and mining

To directly address Gap G3 (Table 4), we assembled commit history spanning 19 open-source repositories across three languages, rather than relying on a single pre-packaged benchmark. The Java portion of this corpus reuses the commit-level dataset released with ApacheJIT [17], covering 13 projects maintained by the Apache Software Foundation; the Python and C/C++ portions were mined directly by us using PyDriller [20] and GitPython to walk each repository's commit graph and extract, for every commit, a unique `commit_id`, the commit message, the unified diff, and the twelve JIT change-level metrics described in Section 3. Table 5 summarises the repositories used per language and the resulting commit counts.

For each repository, mining walked the commit history from the most recent commit backward, so that the sample for every project is a contiguous, recent slice of that project's history rather than a random or uniformly-spaced sample - a deliberate choice that keeps the chronological split described in Section 5 meaningful, since a

Table 5: Open-source repositories used for the multi-language commit dataset.

Language	Repositories used	Commits used
Java	13 Java repositories maintained by the Apache Software Foundation which includes activemq, camel, cassandra, flink, groovy, hadoop, hbase, hive, ignite, kafka, spark, zeppelin, zookeepers	36,000
Python	Django, Flask, CPython	11,996
C/C++	Godot Engine, Krita, OpenCV	12,000
Total	**19 repositories, 3 languages**	**59,996**

strictly chronological split is only informative if the commits genuinely span a continuous and reasonably recent period. A commit was considered part of a project's slice once the target commit count for that language (Table 5) was reached. During post-mining validation, four commits (all C/C++) were found to contain corrupted diff content - a mix of binary artefacts and null bytes that could not be safely tokenised for CodeBERT - and were removed rather than repaired, yielding a final corpus of **59,996 commits**.

4.2 SZZ-based bug labelling

Every mined commit is labelled using the SZZ algorithm [3]: a commit is first flagged as a candidate bug-fix by keyword matching on its commit message (`fix`, `bug`, `error`, `issue`, `patch`, `resolve`, `hotfix`), and for each flagged fix, `git blame` is used to trace the modified lines back to the commit that last touched them - that originating commit is labelled `buggy=1`. All other commits are labelled `buggy=0`. This labelling was applied uniformly across all three languages and all 19 repositories, so that Java, Python, and C/C++ commits share exactly the same bug-inducing-commit definition.

4.3 Schema unification and validation

All mined commits, regardless of source repository or language, are stored in one common 20-column schema (`commit_id, project, buggy, fix, year, author_date, la, ld, nf, nd, ns, ent, ndev, age, nuc, aexp, arexp, asexp, message, diff`), with `project` recorded as a canonical `ecosystem/project` label (e.g. `python/django`, `cpp/krita`) so that language and repository are both recoverable from a single field. Every merge step in the mining pipeline is implemented as a fail-closed validator that checks column-name and column-count consistency, null/empty-value rules on identifier and metric fields, valid binary labels for `buggy/fix`, parseable dates, and the absence of duplicate commit IDs both within and across repositories - aborting without writing output on any anomaly rather than merging silently. The final dataset passed all of these checks with zero duplicate commit IDs and zero duplicate rows.

4.4 CodeBERT semantic embedding

The concatenated message and diff for every one of the 59,996 commits is tokenised and encoded with `microsoft/codebert-base` [4] (message + `[SEP]` + diff, truncated to a fixed token budget, `[CLS]`-token pooling), producing a 768-dimensional embedding per commit. Principal Component Analysis is then fitted *only* on the 41,998 training-split embeddings and applied unchanged to the validation and test splits, reducing the representation to 384 components while retaining 99.43% of the training-set variance - this ordering is what prevents information from validation or test-split commits leaking into the projection itself.

4.5 LLM reasoning feature extraction

Extracting the seven-field reasoning representation (Section 3, Stage 2c) for all 59,996 commits required navigating a hard GPU-time budget on a shared NVIDIA A100 cluster exposed to the notebook only through Multi-Instance GPU (MIG) slices. We summarise the model-selection process here as a reproducibility record; the numeric throughput figures were measured on our specific MIG allocation and are not intended as general hardware benchmarks.

Qwen/Qwen3-8B [21] was the initial candidate and was made to work with 4-bit NF4/BF16 quantization [22], double quantization, and KV-caching enabled, but even after these optimisations its sequential throughput was approximately 600-615 commits/hour, which was incompatible with the available GPU window. Qwen/Qwen3-1.7B was then tested as a smaller, faster alternative; two concurrent 1.7B workers on separate MIG slices reached roughly 1,000 commits/hour, while larger batch sizes (2 prompts/GPU) and shorter generation budgets (100 tokens) were also tested and rejected because they degraded structured-output validity to 0/4 in small-scale tests despite improving nominal throughput. The production configuration ultimately used to generate the released 59,996-row feature set switched to Qwen/Qwen3-4B with 4-bit NF4/BF16 quantization, no double quantization, and a 2048-token input budget. Rather than generating free-form JSON and parsing it, the pipeline scored the seven allowed categorical labels directly from next-token logits, avoiding free-text generation and its associated truncation and malformed-output failure modes. Extraction was checkpointed and resumable by `commit_id`. The final feature file (`llm_features.csv`, 59,996 rows $\times$ 24 columns, 0 duplicate IDs, 0 missing values) stores only the selected label, confidence, and margin for each field, not the raw diff, prompt, or hidden reasoning trace, and is joined to the JIT and CodeBERT tables by `commit_id` prior to modelling. The seven categorical fields are one-hot encoded (never ordinally, to avoid implying a false ordering between, e.g., risk levels) using an encoder fitted only on the training split, while the fourteen confidence/margin scores are retained as continuous features, yielding 51 LLM-derived input columns.

4.6 Final dataset and split

The final multi-language dataset comprises 59,996 commits across 19 repositories and 3 languages, with 0 duplicated commit IDs and 0 duplicated rows. We perform a project-wise chronological split of parsed `author_date` with 70% train / 15% validation / 15%

test to ensure that later commits are always held out for evaluation, and that no future information is leaked into training. This yields 41,998 training, 8,998 validation, and 9,000 test commits (Table 6) with buggy proportion falling from 28.36% in training to 15.29% in the test split, which reflects a genuine temporal drift in the underlying repositories rather than an artefact of resampling (no resampling was applied at any stage).

Table 6: Project-wise chronological split of the 59,996-commit multi-language dataset.

Split	Rows	Buggy	Non-buggy	Buggy %	Non-buggy %
Train	41,998	11,912	30,086	28.36%	71.64%
Validation	8,998	2,082	6,916	23.14%	76.86%
Test	9,000	1,376	7,624	15.29%	84.71%

5 Experimental Setup

Table 7 summarises the common protocol used across all seven experiments. Because the test split is strongly imbalanced toward non-buggy commits (84.71% non-buggy), accuracy alone is not a reliable indicator of bug-detection quality; we therefore report precision, recall, F1, Matthews Correlation Coefficient (MCC), ROC-AUC, and PR-AUC alongside accuracy, and treat F1/MCC/PR-AUC as the primary metrics for comparing configurations.

Seven experiment configurations were run in total, covering every non-trivial subset of the three feature streams (Eq. 1):

1. **Experiment 1 - JIT only.** The 12 standardised JIT metrics, Random Forest and XGBoost.
2. **Experiment 2 - CodeBERT only.** The full 384-D PCA-reduced CodeBERT representation, Random Forest and XGBoost.
3. **Experiment 3 - LLM only.** The 51-D one-hot + confidence/margin LLM representation, Random Forest and XGBoost. This experiment was re-run after a refinement to the LLM feature-encoding pipeline, and the figures reported in Section 6 reflect this final run.
4. **Experiment 4 - JIT + CodeBERT fusion.** Six fusion strategies were compared: (4A) direct 396-feature early concatenation; (4B1/4B2) reduced fusion keeping only the leading 25 or 50 CodeBERT principal components (37 or 62 total features); (4C) mutual-information selection of the top 25 features from the 396-feature early-fusion space, computed on training data only; (4D) probability stacking, where four base learners (JIT-RF, JIT-XGBoost, CodeBERT-RF, CodeBERT-XGBoost) produce validation-split probabilities that train a logistic-regression meta-model; (4E) weighted probability blending, $P_{\text{blend}} = \alpha P_{\text{JIT}} + (1 - \alpha) P_{\text{CodeBERT}}$, with α swept in steps of 0.05 and selected on validation F1; and (4F) a three-model weighted

Table 7: Common experimental configuration used across all seven experiments.

Item	Configuration
Task	Multilingual JIT software defect prediction
Dataset	59,996 commits across Java, Python, and C/C++
Split	Project-wise chronological, 70%/15%/15%
Split sizes	41,998 train / 8,998 validation / 9,000 test
JIT input	12 features: `la, ld, nf, ns, nd, ent, ndev, age, nuc, aexp, arexp, asexp`
CodeBERT	384-D PCA embedding; PCA fitted on train only; 99.43% variance retained
LLM input	51 features: 37 one-hot + 14 confidence/margin scores
Missing values	Train-only median imputation
Target	`buggy`, binarised as 0/1
Random Forest	300–500 trees; balanced class weights; `random_state=42`
XGBoost	300–500 estimators; depth 6; LR 0.05; subsample/colsample 0.8; `scale_pos_weight` ≈ 2.53
LightGBM	Used in 4F and Experiments 7-9, with matching class weight
Threshold	0.5
Resampling	None; class weighting only
Metrics	Accuracy, Precision, Recall, F1, MCC, ROC-AUC, PR-AUC

blend of Random Forest, XGBoost, and LightGBM trained on the full 396-feature representation (Eq. 2).

5. **Experiment 5 - JIT + LLM fusion.** Direct concatenation of the 12 JIT and 51 LLM features (63 total), Random Forest and XGBoost.
6. **Experiment 6 - CodeBERT + LLM fusion.** Direct concatenation of the 384 CodeBERT and 51 LLM features (435 total), Random Forest and XGBoost.
7. **Experiment 7 - Full trimodal fusion.** Direct concatenation of all three streams (12 JIT + 384 CodeBERT + 51 LLM = 447 total features), Random Forest, XGBoost, and LightGBM.

All feature-level operations that could see the test (or validation) split - PCA fitting, one-hot encoding, median imputation, mutual-information ranking, and α selection for blending - are fitted exclusively on the training split and then applied, unchanged, to validation and test, so that no experiment's reported test-set numbers benefit from any form of information leakage.

6 Results

All figures below are computed on the strictly held-out, chronologically future 9,000-commit test split; no test-set information was used for model selection, PCA fitting, feature selection, or hyperparameter/threshold tuning at any stage. Each of the seven experiments is presented as its own subsection: for the three single-modality baselines (Experiments 1-3) we report the methodology, test-set metrics, and XGBoost precision-recall/ROC curves; for the four fusion experiments (Experiments 4-7) we

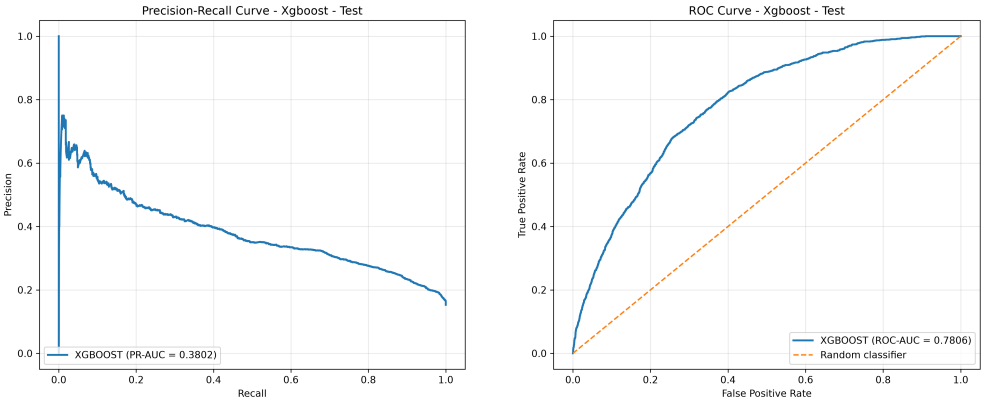
additionally report a comparison table and figure isolating the contribution of the fused stream(s) over their individual single-modality baselines.

6.1 Experiment 1: JIT-Only Baseline

Methodology. The 12 standardised JIT change-level metrics (Table 7) are used as the sole input representation. Random Forest and XGBoost are trained with the configuration in Table 7; no CodeBERT or LLM information is used.

Table 8: Experiment 1 (JIT only) test-set results.

Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
Random Forest	**0.7402**	**0.3178**	0.6097	0.4178	0.2952	0.7681	0.3651
XGBoost	0.6891	0.2949	**0.7427**	**0.4221**	**0.3122**	**0.7806**	**0.3802**



(a) Precision-recall curve

(b) ROC curve

Fig. 3: Experiment 1 (JIT only, XGBoost): test-set precision-recall and ROC curves.

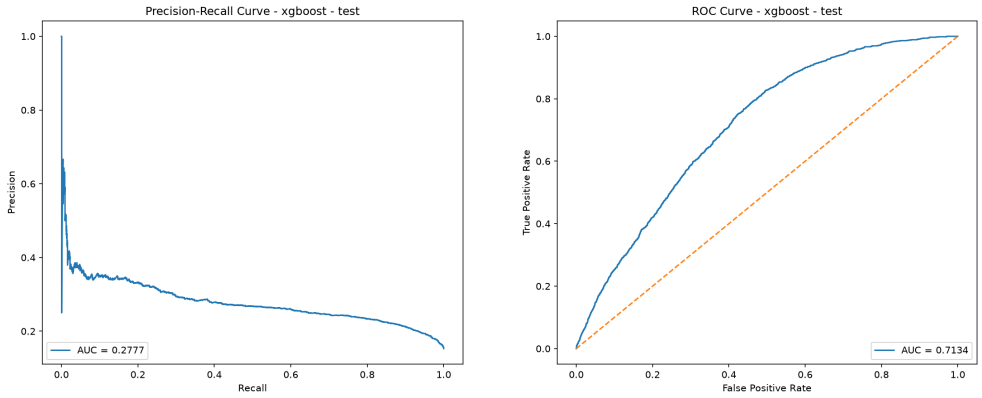
Figure 3 shows the XGBoost precision-recall and ROC curves. XGBoost is the stronger model for this and every subsequent single-modality baseline: Random Forest's higher accuracy (0.7402 vs. 0.6891) comes from a much lower recall (0.6097 vs. 0.7427), i.e. it misses more actual buggy commits while looking better on the imbalanced accuracy metric. On F1, MCC, ROC-AUC, and PR-AUC, XGBoost is uniformly ahead, so we use XGBoost as the reference model for every comparison table in this section unless stated otherwise.

6.2 Experiment 2: CodeBERT-Only Baseline

Methodology. The 384-dimensional PCA-reduced CodeBERT embedding of the commit message and diff (Section 4) is used as the sole input representation, with no JIT or LLM information.

Table 9: Experiment 2 (CodeBERT only) test-set results.

Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
Random Forest	**0.8140**	**0.3307**	0.2115	0.2580	0.1626	0.6987	0.2669
XGBoost	0.6744	0.2584	**0.6039**	**0.3619**	**0.2186**	**0.7134**	**0.2783**



(a) Precision-recall curve

(b) ROC curve

Fig. 4: Experiment 2 (CodeBERT only, XGBoost): test-set precision-recall and ROC curves.

As Table 9 and Figure 4 show, Random Forest's recall collapses to 0.2115 here - it is essentially predicting the majority class for most commits, which is why its accuracy looks strong. On every ranking- and recall-sensitive metric, CodeBERT-only XGBoost (F1 0.3619, ROC-AUC 0.7134) trails JIT-only XGBoost (F1 0.4221, ROC-AUC 0.7806, Table 8), indicating that on this dataset the dense semantic embedding alone is a weaker standalone predictor than the twelve structural metrics, even though (as Section 7 discusses) it still contributes complementary information once fused.

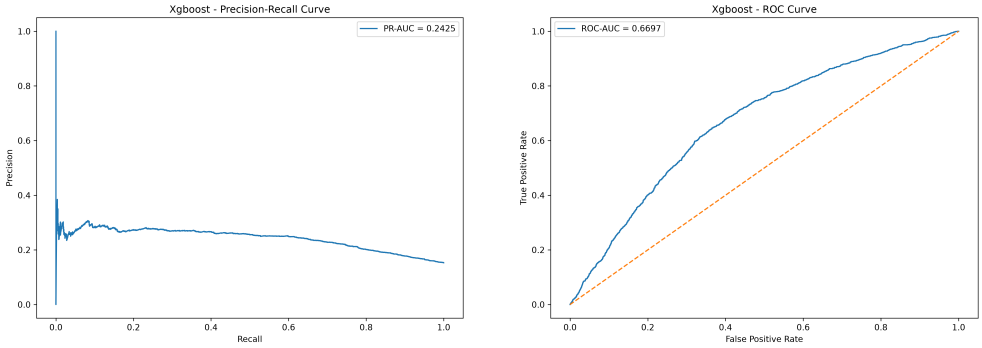
6.3 Experiment 3: LLM-Only Baseline

Methodology. The 51-dimensional LLM reasoning representation (37 one-hot categorical features from the seven structured fields, plus 14 confidence/margin scores) is

used as the sole input representation, with no JIT or CodeBERT information. This experiment was re-conducted after a refinement to the LLM feature-encoding pipeline; the numbers below are the final figures and supersede any earlier draft values.

Table 10: Experiment 3 (LLM only) test-set results.

Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
Random Forest	**0.6881**	**0.2510**	0.5240	0.3394	0.1866	0.6686	0.2401
XGBoost	0.6364	0.2417	**0.6446**	**0.3516**	**0.2048**	**0.6697**	**0.2425**



(a) Precision-recall curve

(b) ROC curve

Fig. 5: Experiment 3 (LLM only, XGBoost, revised run): test-set precision-recall and ROC curves.

Table 10 and Figure 5 show that the Experiment 3 run is the weakest single-modality baseline of the three on F1, MCC, and ROC-AUC, though both models clear the 0.5 no-skill line by a comfortable margin (ROC-AUC 0.6686-0.6697), confirming that the seven-field structured reasoning representation carries genuine, if modest, standalone signal. This is consistent with the representation being deliberately coarse - seven categorical judgements rather than a dense learned embedding - a design choice explained in Section 4 and revisited in Section 7.

6.4 Experiment 4: JIT + CodeBERT Fusion

Methodology. Six different strategies for combining the 12 JIT and 384 CodeBERT features were compared, ranging from simple concatenation to aggressive dimensionality reduction and three probability-level combination schemes (Section 5; 4F is described separately below). Table 11 reports all six.

Table 11: Experiment 4 (JIT + CodeBERT fusion) test-set results across six fusion strategies (4A–4F).

Variant	Model	Feat.	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
4A	Early fusion (RF)	396	**0.7788**	0.3524	0.5334	0.4244	0.3042	0.7772	0.3676
4A	Early fusion (XGB)	396	0.7162	0.3193	0.7565	0.4491	0.3485	0.8037	0.4038
4B1	Reduced fusion, 25 PCA (RF)	37	0.7593	0.3470	0.6512	0.4528	0.3421	0.7965	0.4003
4B1	Reduced fusion, 25 PCA (XGB)	37	0.7088	0.3150	0.7703	0.4472	0.3481	0.8029	**0.4049**
4B2	Reduced fusion, 50 PCA (RF)	62	0.7628	**0.3525**	0.6592	**0.4594**	0.3510	0.7957	0.3952
4B2	Reduced fusion, 50 PCA (XGB)	62	0.7094	0.3166	0.7769	0.4498	0.3524	0.8028	0.3948
4C	Mutual-info top-25 (RF)	25	0.7561	0.3395	0.6294	0.4410	0.3261	0.7881	0.3806
4C	Mutual-info top-25 (XGB)	25	0.6994	0.3067	0.7660	0.4380	0.3359	0.7916	0.3916
4D	Probability stacking (meta-LR)	4	0.6819	0.2962	**0.7856**	0.4302	0.3289	0.7898	0.3766
4E	Weighted blend (RF, α=0.90)	12+384	0.7469	0.3237	0.6017	0.4209	0.2991	0.7706	0.3699
4E	Weighted blend (XGB, α=0.55)	12+384	0.7079	0.3109	0.7485	0.4393	0.3351	0.7896	0.3812
4F	Weighted blend, 3-model (RF+XGB+LGBM)	396	0.7301	0.3290	0.7362	0.4548	**0.3529**	**0.8054**	0.4043

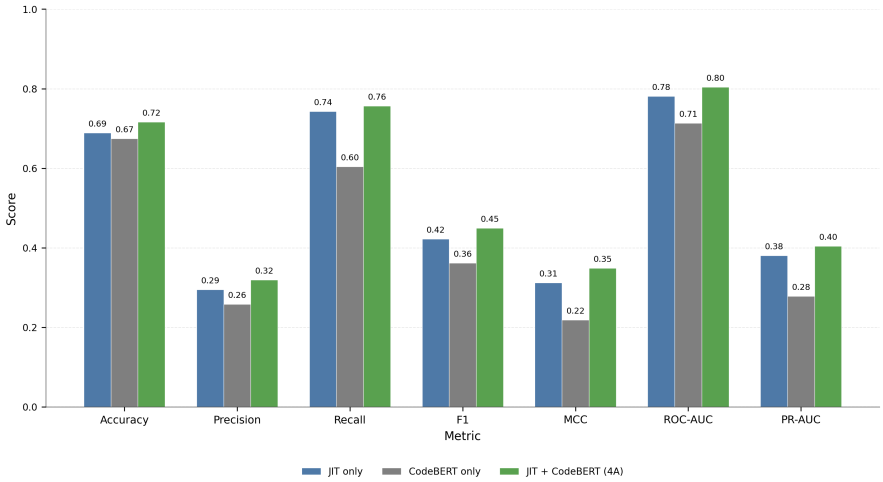


Fig. 6: Experiment 4 comparison: JIT-only vs. CodeBERT-only vs. JIT+CodeBERT early fusion (4A), across all seven metrics.

Table 12 isolates the contribution of fusion itself by comparing the full early-fusion configuration (4A, XGBoost) against the JIT-only and CodeBERT-only baselines it is built from; Figure 6 shows the same comparison graphically.

Table 12: Experiment 4 comparison: JIT-only vs. CodeBERT-only vs. JIT+CodeBERT early fusion (XGBoost).

Metric	JIT only (Exp. 1)	CodeBERT only (Exp. 2)	JIT+CodeBERT (Exp. 4A)
Accuracy	0.6891	0.6744	**0.7162**
Precision	0.2949	0.2584	**0.3193**
Recall	0.7427	0.6039	**0.7565**
F1	0.4221	0.3619	**0.4491**
MCC	0.3122	0.2186	**0.3485**
ROC-AUC	0.7806	0.7134	**0.8037**
PR-AUC	0.3802	0.2783	**0.4038**

Fusion beats both single-modality baselines on every metric we measured. The smallest gain is in recall, where it sits only slightly above JIT-only (0.7565 vs. 0.7427), but the other metrics all improve clearly. This is the most consistent gain of any pairing in this paper, and it directly confirms Gap G1 in Table 4. Two further observations from Table 11: direct feature-level fusion (4A/4B) beats the two simpler probability-level strategies (4D stacking, 4E two-model blending) on F1 and MCC, with the three-model blend 4F (described below) as the one probability-level method that matches or beats them, and the heavily reduced 4B2 configuration (62 features, an 84.3% reduction from 4A) matches or slightly exceeds 4A on F1 and MCC, showing that

most of the useful CodeBERT signal is concentrated in a small number of leading principal components.

6.4.1 Experiment 4F: Three-Model Weighted Probability Blend

A sixth JIT+CodeBERT variant extends the two-model blend of 4E to three base learners - Random Forest, XGBoost, and LightGBM - each trained independently on the full 396-feature (12 JIT + 384 CodeBERT) representation, and combined as

$$P_{\text{blend}} = w_{\text{RF}}\, P_{\text{RF}} + w_{\text{XGB}}\, P_{\text{XGBoost}} + w_{\text{LGBM}}\, P_{\text{LightGBM}}, \qquad w_{\text{RF}} + w_{\text{XGB}} + w_{\text{LGBM}} = 1, \tag{2}$$

with the weights selected on the validation split only (grid search, step 0.05, ranked by F1 then MCC then ROC-AUC then PR-AUC) and then frozen for test evaluation. The selected weights were $w_{\text{RF}} = 0.35$, $w_{\text{XGB}} = 0.30$, $w_{\text{LGBM}} = 0.35$. Table 11 includes this configuration as variant 4F: on test, it reaches F1 0.4548, MCC 0.3529, and ROC-AUC 0.8054. These are the best MCC and ROC-AUC of any JIT+CodeBERT variant in Table 11, and its F1 is topped only by the 4B2 Random Forest (0.4594), which gets there with a much lower recall (0.6592 vs. 0.7362). Its ROC-AUC is marginally above 4A's 0.8037.

6.5 Experiment 5: JIT + LLM Fusion

Methodology. The 12 JIT features and 51 LLM reasoning features are concatenated directly (63 total features), with no CodeBERT information. Median imputation and one-hot encoding are fitted on the training split only, matching the protocol used throughout.

Table 13: Experiment 5 (JIT + LLM fusion) test-set results.

Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
Random Forest	**0.7394**	**0.3161**	0.6054	0.4154	0.2919	0.7692	0.3544
XGBoost	0.6886	0.2949	**0.7456**	**0.4227**	**0.3133**	**0.7780**	**0.3812**

Comparing Table 13 against Table 14 (see also Figure 7), adding the LLM stream to JIT barely moves any metric: F1 improves by only 0.0006, MCC by 0.0011, and ROC-AUC actually falls slightly (by 0.0026) relative to JIT-only. This is a materially weaker fusion effect than JIT+CodeBERT (Table 12), and it is the first indication in this paper's results that the LLM reasoning stream, at least in its current seven-field schema, adds little on top of the JIT metrics specifically - a point we return to in Section 7.

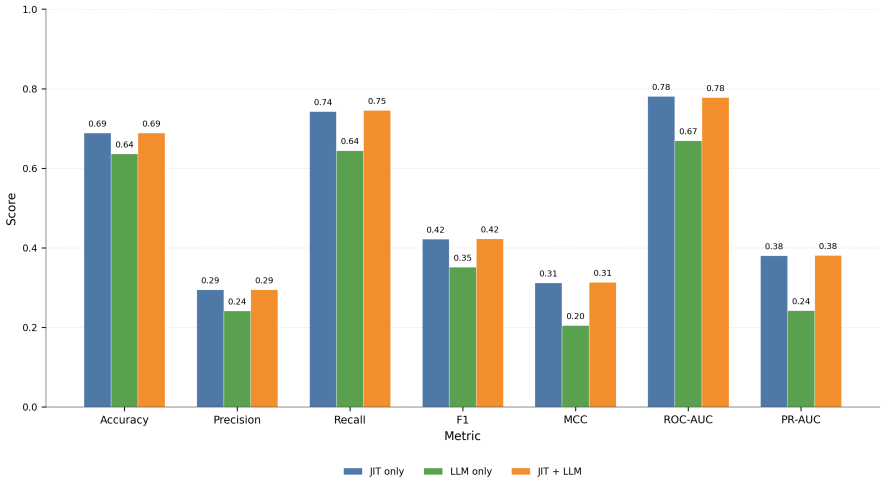


Fig. 7: Experiment 5 comparison: JIT-only vs. LLM-only vs. JIT+LLM fusion, across all seven metrics.

Table 14: Experiment 5 comparison: JIT-only vs. LLM-only vs. JIT+LLM fusion (XGBoost).

Metric	JIT only (Exp. 1)	LLM only (Exp. 3)	JIT+LLM (Exp. 5)
Accuracy	**0.6891**	0.6364	0.6886
Precision	**0.2949**	0.2417	**0.2949**
Recall	0.7427	0.6446	**0.7456**
F1	0.4221	0.3516	**0.4227**
MCC	0.3122	0.2048	**0.3133**
ROC-AUC	**0.7806**	0.6697	0.7780
PR-AUC	0.3802	0.2425	**0.3812**

6.6 Experiment 6: CodeBERT + LLM Fusion

Methodology. The 384 CodeBERT features and 51 LLM reasoning features are concatenated directly (435 total features), with no JIT information. PCA, one-hot encoding, and imputation are all fitted on the training split only.

Table 15: Experiment 6 (CodeBERT + LLM fusion) test-set results.

Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
Random Forest	**0.7610**	**0.3171**	0.4884	0.3845	0.2533	0.7330	0.3045
XGBoost	0.6987	0.2934	**0.6897**	**0.4117**	**0.2925**	**0.7626**	**0.3389**

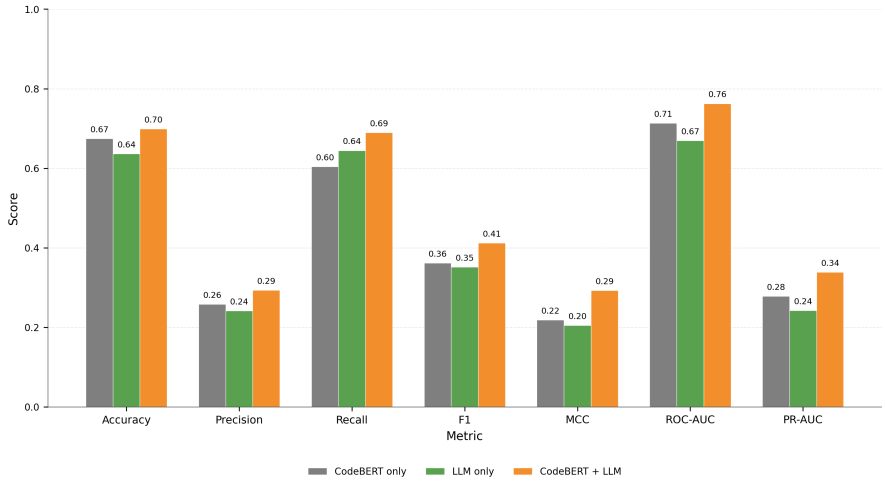


Fig. 8: Experiment 6 comparison: CodeBERT-only vs. LLM-only vs. CodeBERT+LLM fusion, across all seven metrics.

Table 16: Experiment 6 comparison: CodeBERT-only vs. LLM-only vs. CodeBERT+LLM fusion (XGBoost).

Metric	CodeBERT only (Exp. 2)	LLM only (Exp. 3)	CodeBERT+LLM (Exp. 6)
Accuracy	0.6744	0.6364	**0.6987**
Precision	0.2584	0.2417	**0.2934**
Recall	0.6039	0.6446	**0.6897**
F1	0.3619	0.3516	**0.4117**
MCC	0.2186	0.2048	**0.2925**
ROC-AUC	0.7134	0.6697	**0.7626**
PR-AUC	0.2783	0.2425	**0.3389**

Comparing Table 15 against Table 16 (see also Figure 8): unlike its near-negligible effect when paired with JIT (Table 14), the LLM stream meaningfully improves on CodeBERT-only when the two are fused: F1 rises from 0.3619 to 0.4117 (+0.0498), MCC from 0.2186 to 0.2925 (+0.0739), and ROC-AUC from 0.7134 to 0.7626 (+0.0492) – the largest relative gain from adding the LLM stream to any single baseline observed in this paper. This asymmetry (LLM helps CodeBERT more than it helps JIT) is itself a useful empirical clue about what kind of information the LLM stream carries, discussed further in Section 7.

6.7 Experiment 7: Full Trimodal Fusion (JIT + CodeBERT + LLM)

Methodology. All three streams are concatenated directly: 12 JIT + 384 PCA-reduced CodeBERT + 51 LLM reasoning features, for 447 total input features.

Random Forest, XGBoost, and LightGBM are all trained on this representation (LightGBM was already used in the 4F blend), since a full trimodal input is where classifier diversity is most likely to pay off. We call this full-concatenation configuration variant **7A**. Five further variants (7B-7F) built on the same 447-feature space are reported in a later subsection.

Table 17: Experiment 7A (full JIT + CodeBERT + LLM trimodal fusion, direct concatenation) test-set results.

Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
Random Forest	**0.7633**	**0.3403**	0.5836	0.4299	0.3104	0.7736	0.3593
XGBoost	0.7138	0.3176	0.7594	**0.4479**	0.3475	**0.8019**	**0.3985**
LightGBM	0.7022	0.3118	**0.7849**	0.4463	**0.3492**	**0.8028**[1]	0.3984

[1]LightGBM's ROC-AUC (0.8028) is nominally the single highest value in this table, 0.0009 above XGBoost's 0.8019; the two are effectively tied given the granularity of the reported figures.

Since Experiment 7A (Table 17) is the culmination of the fusion sweep, we compare it pairwise against each of the three two-stream fusions that feed into it (Tables 18–20), all using XGBoost as the consistent reference model. Figure 9 additionally compares 7A against the three single-stream baselines..

Table 18: Experiment 7A vs. Experiment 4B2 (best JIT+CodeBERT reduced fusion), XGBoost.

Metric	JIT+CodeBERT (Exp. 4B2)	All three (Exp. 7A)	Change
Accuracy	0.7094	**0.7138**	+0.0044
Precision	0.3166	**0.3176**	+0.0011
Recall	**0.7769**	0.7594	−0.0174
F1	**0.4498**	0.4479	−0.0019
MCC	**0.3524**	0.3475	−0.0049
ROC-AUC	**0.8028**	0.8019	−0.0009
PR-AUC	0.3948	**0.3985**	+0.0037

The pattern across Tables 18–20 is consistent and important: adding the missing stream to Experiment 5 (JIT+LLM → full fusion, Table 19) or Experiment 6 (CodeBERT+LLM → full fusion, Table 20) produces a large, unambiguous improvement on every metric, because in both cases the missing stream is CodeBERT or JIT - the two streams already shown to fuse strongly together in Experiment 4. But adding LLM to the already-strong JIT+CodeBERT pair (Table 18) produces a slightly *mixed* result: PR-AUC and precision improve marginally, but recall, F1, MCC, and ROC-AUC all move slightly backward relative to Experiment 4B2, despite Experiment 7A using more than seven times as many features (447 vs. 62).

Table 19: Experiment 7A vs. Experiment 5 (JIT+LLM), XGBoost.

Metric	JIT+LLM (Exp. 5)	All three (Exp. 7A)	Change
Accuracy	0.6886	**0.7138**	+0.0252
Precision	0.2949	**0.3176**	+0.0227
Recall	0.7456	**0.7594**	+0.0138
F1	0.4227	**0.4479**	+0.0252
MCC	0.3133	**0.3475**	+0.0342
ROC-AUC	0.7780	**0.8019**	+0.0239
PR-AUC	0.3812	**0.3985**	+0.0173

Table 20: Experiment 7A vs. Experiment 6 (CodeBERT+LLM), XGBoost.

Metric	CodeBERT+LLM (Exp. 6)	All three (Exp. 7A)	Change
Accuracy	0.6987	**0.7138**	+0.0151
Precision	0.2934	**0.3176**	+0.0242
Recall	0.6897	**0.7594**	+0.0698
F1	0.4117	**0.4479**	+0.0362
MCC	0.2925	**0.3475**	+0.0550
ROC-AUC	0.7626	**0.8019**	+0.0394
PR-AUC	0.3389	**0.3985**	+0.0596

6.8 Experiment 7 Variants (7B-7F)

Five further variants were run on top of the Experiment 7A feature space to test whether a different combination strategy, a reduced CodeBERT representation, or explicit feature selection could improve on 7A's result: (7B) JIT + the leading 25 CodeBERT PCA components + LLM (88 features); (7C) JIT + the leading 50 CodeBERT PCA components + LLM (113 features); (7D) mutual-information selection of the top 25 features from the full 447-feature space, fitted on training data only; (7E) probability stacking of the 7A base models' (Random Forest, XGBoost, LightGBM) validation-split probabilities through a logistic-regression meta-model; and (7F) a three-model weighted probability blend of the same three base models (Eq. 2), with weights selected on validation only. Table 21 reports test-set results for all five, and Figure 10 compares F1, MCC, ROC-AUC, and PR-AUC across 7A-7F.

None of the five variants clearly surpasses 7A: 7F (weighted 3-model blend) edges out 7A's XGBoost result on F1 (0.4519 vs. 0.4479) and MCC (0.3535 vs. 0.3475), and 7B and 7C reach a broadly similar F1/MCC band to 7A despite using far fewer CodeBERT dimensions, but no variant produces a decisive improvement over the 4A/4B2/7A/7F cluster that has defined the ceiling of this paper's results throughout Section 6. 7D, which restricts the model to only 25 mutual-information-selected

Table 21: Experiment 7 variants (7B–7F) test-set results, alongside 7A for reference.

Variant	Model	Feat.	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
7A	Full early fusion (RF)	447	**0.7633**	0.3403	0.5836	0.4299	0.3104	0.7736	0.3593
7A	Full early fusion (XGB)	447	0.7138	0.3176	0.7594	0.4479	0.3475	0.8019	0.3985
7A	Full early fusion (LGBM)	447	0.7022	0.3118	0.7849	0.4463	0.3492	0.8028	0.3984
7B	JIT + CodeBERT-25 + LLM (RF)	88	0.7582	**0.3444**	0.6432	0.4486	0.3364	0.7900	0.3821
7B	JIT + CodeBERT-25 + LLM (XGB)	88	0.7019	0.3095	0.7718	0.4419	0.3416	0.7988	0.3932
7B	JIT + CodeBERT-25 + LLM (LGBM)	88	0.6942	0.3052	0.7834	0.4393	0.3402	0.7978	0.3881
7C	JIT + CodeBERT-50 + LLM (RF)	113	0.7521	0.3348	0.6294	0.4370	0.3210	0.7880	0.3790
7C	JIT + CodeBERT-50 + LLM (XGB)	113	0.7012	0.3072	0.7602	0.4376	0.3345	0.7989	0.3997
7C	JIT + CodeBERT-50 + LLM (LGBM)	113	0.6928	0.3036	0.7805	0.4372	0.3371	0.8000	0.4014
7D	MI top-25 fusion (RF)	25	0.7423	0.3225	0.6228	0.4250	0.3051	0.7719	0.3616
7D	MI top-25 fusion (XGB)	25	0.6871	0.2950	0.7529	0.4239	0.3159	0.7814	0.3825
7D	MI top-25 fusion (LGBM)	25	0.6808	0.2927	0.7682	0.4239	0.3181	0.7832	0.3826
7E	Probability stacking (meta-LR)	3	0.6998	0.3105	**0.7892**	0.4456	0.3491	0.8041	0.3966
7F	Weighted blend (RF+XGB+LGBM)	447	0.7157	0.3204	0.7667	**0.4519**	**0.3535**	**0.8049**	**0.4052**

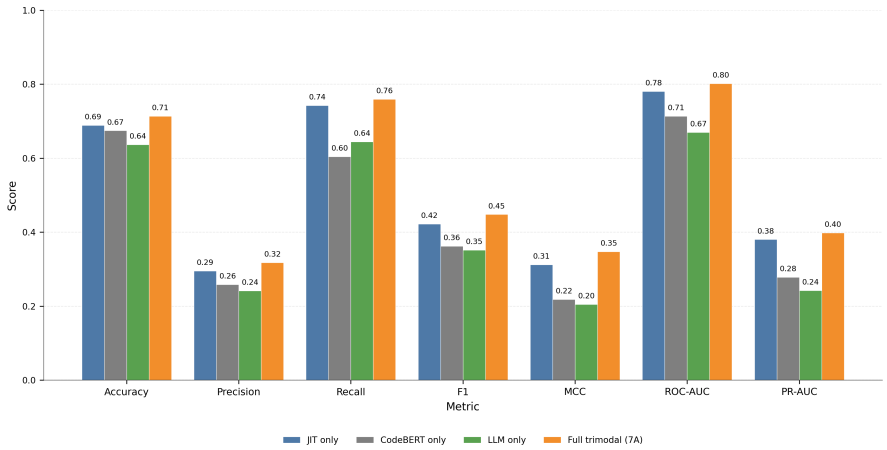


Fig. 9: Experiment 7A comparison: JIT-only vs. CodeBERT-only vs. LLM-only vs. full trimodal fusion (7A), across all seven metrics.

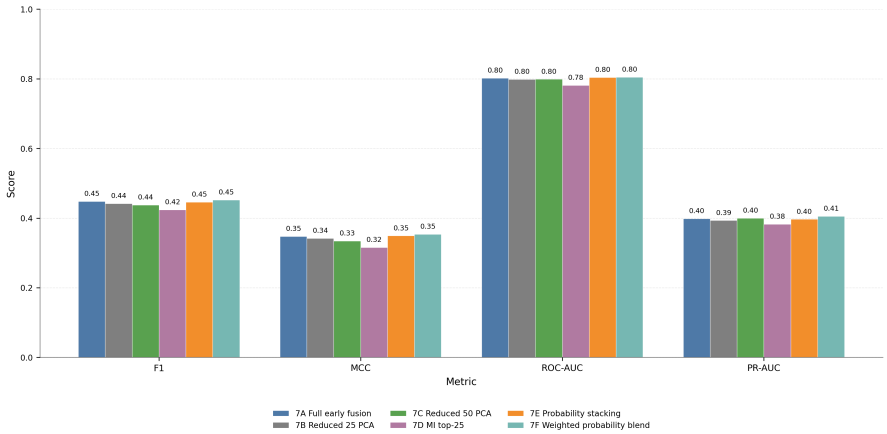


Fig. 10: Comparison of F1, MCC, ROC-AUC, and PR-AUC across the six Experiment 7 variants (7A–7F).

features, gives up a small amount of F1 and MCC relative to the other variants - consistent with aggressive feature selection discarding some usable signal even when it is fitted correctly on training data only.

Table 22 lists the 25 features selected by 7D's mutual-information ranking. The composition is uneven across modalities: 11 of the 12 JIT features are selected (all except one), 13 of the 51 LLM features (mostly confidence and margin scores rather than the one-hot category labels themselves), and only a single CodeBERT component (`codebert_pca_1`) out of 384. Mutual information, computed independently per

feature, favours the JIT metrics and the LLM confidence/margin scores over individual CodeBERT PCA components - which is expected, since any single PCA dimension carries only a small share of the variance captured jointly across all 384 components, so no individual CodeBERT feature ranks highly in a per-feature univariate criterion even though the representation as a whole is informative (as Experiments 2, 4, 6, and 7A-7C all show).

Table 22: Experiment 7D: the 25 features selected by training-only mutual information, by modality.

Modality	Count	Selected features
JIT	11	`jit_la, jit_age, jit_nf, jit_ent, jit_ld, jit_nd, jit_asexp, jit_ns, jit_arexp, jit_ndev, jit_nuc`
CodeBERT	1	`codebert_pca_1`
LLM	13	`complexity_confidence, complexity_margin, complexity_M, scope_margin, risk_margin, risk_confidence, scope_M, scope_confidence, risk_M, test_M, test_confidence, security_margin, security_confidence`

For 7F, the same weighted-blend formulation as 4F (Eq. 2) is applied to the three Experiment 7A base models rather than the two Experiment 4A/4F base models, with validation-selected weights $w_{\mathrm{RF}} = 0.15$, $w_{\mathrm{XGB}} = 0.35$, $w_{\mathrm{LGBM}} = 0.50$ – markedly different from 4F's more balanced $(0.35, 0.30, 0.35)$, reflecting that once the LLM stream is present in the base features, the validation set favours weighting LightGBM's predictions most heavily.

6.9 Consolidated cross-experiment comparison

Table 23 places all seven base experiments side by side using XGBoost throughout, the consistently stronger single-model family across every configuration in this paper, together with the two best-performing multi-model blends (4F, 7F), giving a single view of the complete fusion sweep.

Two conclusions follow directly from Table 23. First, every fusion configuration that includes both JIT and CodeBERT (4A, 4B2, 4F, 7A, 7F) clearly outperforms every configuration that does not (1, 2, 3, 5, 6) on F1, MCC, and ROC-AUC, reinforcing JIT+CodeBERT as the load-bearing pairing in this framework. Second, and more surprisingly, the full trimodal fusion does not pull clearly ahead of the best JIT+CodeBERT configurations. 7F has the highest MCC in the table (0.3535), but that is only 0.0006 above 4F (0.3529), and 4F is actually ahead on F1 (0.4548 vs. 0.4519) and ROC-AUC (0.8054 vs. 0.8049). 4B2 is also within about 0.002 of 7F on both F1 and MCC while using 62 features instead of 447. In other words, adding the LLM stream on top of an already strong bimodal representation gave us, at best, a very small and inconsistent gain. We look into the reason in the diagnosis experiments below.

Table 23: Consolidated test-set comparison across all seven experiments (XGBoost, plus the two best multi-model blends).

Exp.	Configuration	Acc.	F1	MCC	ROC-AUC	PR-AUC
1	JIT only	0.6891	0.4221	0.3122	0.7806	0.3802
2	CodeBERT only	0.6744	0.3619	0.2186	0.7134	0.2783
3	LLM only	0.6364	0.3516	0.2048	0.6697	0.2425
4A	JIT + CodeBERT	0.7162	0.4491	0.3485	0.8037	0.4038
4B2	JIT + CodeBERT (reduced, 62 feat.)	0.7094	0.4498	0.3524	0.8028	0.3948
4F	JIT + CodeBERT (3-model blend)	**0.7301**	**0.4548**	0.3529	**0.8054**	0.4043
5	JIT + LLM	0.6886	0.4227	0.3133	0.7780	0.3812
6	CodeBERT + LLM	0.6987	0.4117	0.2925	0.7626	0.3389
7A	JIT + CodeBERT + LLM	0.7138	0.4479	0.3475	0.8019	0.3985
7F	JIT + CodeBERT + LLM (3-model blend)	0.7157	0.4519	**0.3535**	0.8049	**0.4052**

6.10 Diagnosis: Why Trimodal Fusion Does Not Overtake JIT+CodeBERT (Experiments 8-9)

The consolidated result above raises a specific question: does the LLM stream fail to move the trimodal result past 4F because the LLM stream itself carries little signal, or because the 51-feature LLM representation is internally redundant, so that most of its 51 dimensions add noise rather than information once combined with the other two streams? Experiments 8 and 9 were designed to distinguish between these two explanations.

6.10.1 Diagnostic analysis of the 51-feature LLM representation

Before running either experiment, we profiled the 51-feature LLM representation on the training split only. Three findings motivated the reduction that follows. First, a univariate signal check found that the strongest individual predictors within the LLM stream were concentrated in the confidence and margin scores rather than the one-hot category labels themselves, most notably `complexity_confidence`, `complexity_margin`, `risk_confidence`, `security_confidence`, `scope_confidence`, and `scope_margin`; several individual categorical indicators carried weaker signal in comparison. Second, a Cramér's V analysis of the seven categorical dimensions found substantial pairwise dependence rather than independence - most strongly between risk and security ($V \approx 0.7396$), with further sizeable dependence between complexity and scope, change and complexity, change and scope, risk and test, and complexity and test. Third, the training-set buggy rate varied considerably across categories within a single dimension: for example, intent category BF had a buggy rate of approximately 19.84% against 42.10% for PF, and change category OT had a buggy rate of approximately 19.21% against 48.07% for UI. Together, these three observations indicate that the LLM stream carries genuine defect-related signal, but that its 51 model-ready features are neither independent nor uniformly informative - exactly the precondition under which a large one-hot categorical block can behave as redundant

input once combined with a 384-dimensional CodeBERT embedding, consistent with the interpretation in Section 7.

6.10.2 Experiment 8: Compact confidence/margin LLM representation

Methodology. Experiment 8 keeps only the 14 confidence and margin scores (two per semantic dimension, across the seven dimensions) and drops the 37 one-hot categorical columns entirely, combined with the unchanged 12 JIT metrics and 384 train-only-PCA CodeBERT features, for 410 total model features - a 72.55% reduction in LLM dimensionality relative to the 51-feature representation used in Experiments 6 and 7. Random Forest, XGBoost, and LightGBM are trained as in Experiment 7, and a validation-selected weighted blend (Eq. 2) is computed with weights $w_{\mathrm{RF}} = 0.15$, $w_{\mathrm{XGB}} = 0.40$, $w_{\mathrm{LGBM}} = 0.45$.

6.10.3 Experiment 9: Redundancy-aware LLM feature selection

Methodology. Experiment 9 asks whether an explicitly redundancy-aware selection of 14 LLM features can do better than simply keeping the confidence/margin block. Starting from the full 51-feature LLM representation (fitted on training data only, as throughout this paper), mutual information with the buggy label is computed per feature on the training split, and features are greedily removed whenever their absolute Pearson correlation with an already-retained, higher-MI feature reaches or exceeds 0.90. This redundancy filter reduces the 51 candidates to 44 non-redundant features, from which the top 14 by training mutual information are kept, combined with the same 12 JIT and 384 CodeBERT features for 410 total model features. The 14 selected features are listed in Table 24; six of the fourteen are confidence or margin scores (`complexity_margin`, `risk_confidence`, `scope_margin`, `test_confidence`, `security_margin`, and `change_confidence`), while the other eight are one-hot category indicators (`scope_M`, `complexity_M`, `test_M`, `risk_M`, `security_N`, `change_OT`, `complexity_L`, and `test_L`), which Experiment 8's fixed confidence/margin selection does not include. Model training and blend-weight selection follow the same protocol as Experiment 8; the validation-selected weights were $w_{\mathrm{RF}} = 0.10$, $w_{\mathrm{XGB}} = 0.80$, $w_{\mathrm{LGBM}} = 0.10$.

Table 24: Experiment 9: the 14 LLM features selected by train-only mutual-information ranking after correlation-based redundancy filtering ($|r| \geq 0.90$).

Selected features
`complexity_margin, scope_M, risk_confidence, scope_margin, complexity_M, test_M, test_confidence, security_margin, security_N, change_confidence, risk_M, change_OT, complexity_L, test_L`

As Table 25 shows, Experiment 8 outperforms Experiment 9 on every one of F1, MCC, ROC-AUC, and PR-AUC at the blended level, though the margins are small

Table 25: Experiment 8 and Experiment 9 test-set results (12 JIT + 384 CodeBERT + 14 LLM = 410 features).

Exp.	Model	Acc.	Prec.	Recall	F1	MCC	ROC-AUC	PR-AUC
8	Random Forest	0.7650	0.3398	0.5698	0.4257	0.3050	0.7741	0.3637
8	XGBoost	0.7172	0.3214	0.7645	0.4526	0.3540	0.8026	0.4005
8	LightGBM	0.7006	0.3104	**0.7849**	0.4449	0.3475	0.8034	0.3993
8	Weighted blend	0.7180	0.3223	0.7660	**0.4537**	**0.3557**	**0.8046**	**0.4027**
9	Random Forest	**0.7686**	**0.3460**	0.5770	0.4326	0.3139	0.7751	0.3589
9	XGBoost	0.7144	0.3190	0.7645	0.4502	0.3510	0.8022	0.3951
9	LightGBM	0.7007	0.3092	0.7762	0.4422	0.3428	0.8027	0.3989
9	Weighted blend	0.7181	0.3219	0.7624	0.4526	0.3538	0.8031	0.3958

(F1 0.4537 vs. 0.4526, MCC 0.3557 vs. 0.3538). This does not support a claim that explicit mutual-information-plus-correlation selection finds a better 14-feature subset than simply keeping the confidence/margin block; the more defensible reading is that once the LLM representation has been reduced to a compact, low-redundancy size, the exact choice of which 14 features to keep matters much less than the act of reducing from 51 to 14 in the first place.

Table 26 states this reduction as an explicit difference: going from the full 51-feature LLM representation (Experiment 7F) to the 14-feature confidence/margin representation (Experiment 8) while holding the JIT and CodeBERT streams fixed.

Table 26: Effect of reducing the LLM stream from 51 to 14 confidence/margin features, holding JIT and CodeBERT fixed (weighted blend).

Metric	51 LLM features (Exp. 7F)	14 LLM features (Exp. 8)	Change
Accuracy	0.7157	**0.7180**	+0.0023
Precision	0.3204	**0.3223**	+0.0019
Recall	**0.7667**	0.7660	−0.0007
F1	0.4519	**0.4537**	+0.0018
MCC	0.3535	**0.3557**	+0.0022
ROC-AUC	**0.8049**	0.8046	−0.0003
PR-AUC	**0.4052**	0.4027	−0.0025

Every change in Table 26 is within 0.0025 of zero, and four of the seven metrics (accuracy, precision, F1, and MCC) actually improve slightly rather than getting worse when 37 LLM columns are dropped. This is the direct evidence that the 51-feature representation was carrying redundant, not uniquely useful, information: removing most of it does not cost performance, and on the metric this paper treats as most diagnostic of balanced classification quality (MCC), it helps.

Table 27 states the second comparison as an explicit difference: holding the reduced 14-feature budget fixed, does deliberately selecting *which* 14 features to keep (Experiment 9's mutual-information-plus-correlation procedure) do better than simply keeping the 14 confidence/margin scores (Experiment 8)?

Table 27: Effect of replacing the fixed confidence/margin 14-feature LLM subset (Exp. 8) with the MI/correlation-selected 14-feature subset (Exp. 9), holding JIT and CodeBERT fixed (weighted blend).

Metric	Confidence/margin 14 (Exp. 8)	MI/correlation-selected 14 (Exp. 9)	Change
Accuracy	0.7180	**0.7181**	+0.0001
Precision	**0.3223**	0.3219	−0.0004
Recall	**0.7660**	0.7624	−0.0036
F1	**0.4537**	0.4526	−0.0011
MCC	**0.3557**	0.3538	−0.0019
ROC-AUC	**0.8046**	0.8031	−0.0015
PR-AUC	**0.4027**	0.3958	−0.0069

Unlike Table 26, every change in Table 27 except accuracy is negative: explicit redundancy-aware selection does not improve on the simpler confidence/margin subset, and on PR-AUC it gives up the largest margin observed anywhere in this diagnosis (−0.0069). The reduction from 51 to 14 features is therefore what matters, not the particular selection rule used to choose which 14 to keep.

Table 28: Direct comparison of the weighted-blend result for 4F, 7F, 8, and 9.

Exp.	JIT	CodeBERT	LLM	Total	Acc.	F1	MCC	ROC-AUC
4F	12	384	0	396	**0.7301**	**0.4548**	0.3529	**0.8054**
7F	12	384	51	447	0.7157	0.4519	0.3535	0.8049
8	12	384	14 (confidence/margin)	410	0.7180	0.4537	**0.3557**	0.8046
9	12	384	14 (MI/correlation-selected)	410	0.7181	0.4526	0.3538	0.8031

Table 28 puts the diagnosis in context. Cutting the LLM stream from 51 to 14 confidence/margin features (Experiment 8) gives back essentially everything Experiment 7F achieved: F1 is actually slightly higher (0.4537 vs. 0.4519) and PR-AUC is within 0.0025, with 37 fewer LLM columns. This confirms that the 51-feature representation used in Experiments 6, 7A, and 7B–7F carried a lot of redundant information, though it was not actively harmful, which fits the strong dependence between categories reported above. At the same time, neither Experiment 8 nor Experiment 9 overtakes 4F on F1 or ROC-AUC, so a compact LLM representation does not close the gap to the strongest JIT+CodeBERT configuration. The LLM stream is not without value here, though. Experiment 8's MCC (0.3557) is the highest we obtained anywhere

in this paper, ahead of 7F (0.3535) and 4F (0.3529), and its recall (0.7660) is higher than 4F's (0.7362). Its PR-AUC (0.4027), however, is marginally below 4F's (0.4043). The honest summary is that the LLM stream gives a small lift in MCC and recall over 4F, while F1, ROC-AUC, and PR-AUC stay essentially where 4F already had them, and most of that small lift can be obtained from a 14-feature confidence/margin summary instead of the full 51-feature representation. All of these differences are small and come from a single split, so they should be read as indications rather than firm conclusions.

7 Discussion

7.1 Why JIT+CodeBERT fusion works so consistently

The gain from JIT+CodeBERT fusion (Table 12) is compatible with the motivation in Section 1 and Gap G1 in Table 4: JIT metrics describe *how much* changed and by whom, while CodeBERT describes *what* changed at the token level, and a tree ensemble given both jointly can split on combinations neither stream offers alone - for instance, a small diff, by JIT metrics, that CodeBERT nonetheless represents as unusual relative to the developer's typical change. This is the strongest and most reliable pairing observed anywhere in this paper's results, and it holds regardless of whether the CodeBERT side is expressed as the full 384-D embedding, a heavily reduced PCA subset, or a mutual-information-selected feature list (Table 11).

7.2 The LLM stream is asymmetrically useful

A pattern only visible once all pairwise experiments are compared side by side (Tables 14 and 16) is that the LLM reasoning stream helps CodeBERT far more than it helps JIT: fused with CodeBERT, F1 rises by 0.0498 and MCC by 0.0739 (Experiment 6 vs. Experiment 2); fused with JIT, F1 rises by only 0.0006 and MCC by 0.0011 (Experiment 5 vs. Experiment 1). A plausible explanation is that the LLM's seven categorical fields (intent, risk, complexity, scope, test impact, security risk, change type) are themselves a coarse, discretised summary of the same code-level content CodeBERT already encodes densely - so pairing the LLM stream with CodeBERT adds a complementary, differently-shaped view of similar underlying information, whereas pairing it with the purely structural JIT metrics adds comparatively little the JIT metrics cannot already infer from change size and developer history. If this explanation is correct, it also predicts the central finding of Section 6: once JIT and CodeBERT are already fused (Experiment 4), the LLM stream's remaining marginal contribution is small, because most of what it would add is already present via CodeBERT.

7.3 Why full trimodal fusion does not clearly beat the best bimodal fusion

Experiment 7 (Table 23) is competitive with, but does not surpass, the best JIT+CodeBERT-only configurations, despite using 447 features against 62-396. Taken together with Section 7's previous observation, the most parsimonious explanation is

redundancy rather than noise: the LLM stream's seven-field schema was deliberately kept coarse for extraction tractability (Section 4), and once CodeBERT's dense 384-D representation is already present, the LLM fields appear to contribute little that is not already recoverable from CodeBERT, while still adding 51 dimensions of essentially redundant one-hot-encoded categorical noise for the tree ensembles to split around. Experiments 8 and 9 (Section 6) test this redundancy hypothesis directly: reducing the LLM stream from 51 to 14 confidence/margin features recovers essentially all of the full LLM representation's contribution to F1 and PR-AUC, and a more deliberate mutual-information-plus-correlation selection of a different 14-feature subset does not do any better, confirming that the 51-feature representation's ceiling was limited by internal redundancy rather than by an insufficient number of features. This does not mean the LLM stream is uninformative in general - it is clearly informative on its own (Table 10), it produces a large gain when paired with CodeBERT (Table 16), and the compact 14-feature version in Experiment 8 achieves the highest MCC of any configuration reported in this paper (0.3557), with higher recall than 4F (0.7660 vs. 0.7362). What it does not do, under any of the LLM schemas we tested, is push F1, ROC-AUC, or PR-AUC meaningfully past what JIT+CodeBERT already achieves on this benchmark.

7.4 Feature-count efficiency

The practical implication compounds across Sections 7.2 and 7.3: the 62-feature reduced JIT+CodeBERT fusion (4B2) matches or exceeds the 447-feature full trimodal fusion (Experiment 7) on F1 and MCC, and does so with roughly one-seventh the input dimensionality. For a framework whose intended deployment is real-time CI/CD gating, this is a meaningful practical finding independent of the trimodal question: most of the usable signal in this framework is concentrated in a small number of JIT metrics plus a small number of leading CodeBERT principal components, and neither the full 384-D CodeBERT embedding nor the full 51-D LLM representation is necessary to reach near-best performance.

7.5 Engineering lessons from fractional-GPU LLM inference

Beyond the classifier results, running the LLM reasoning extraction stage to completion required treating GPU capacity as an empirical quantity to be measured rather than a nominal specification to be trusted. Our initial MIG allocation nominally exposed multiple ∼5 GB slices, but per-slice free memory measured by direct allocation stress tests ranged from effectively out-of-memory to ∼4.6 GB free on ostensibly identical instances, and container-level permission restrictions prevented us from identifying the owning processes for the unavailable capacity from inside the notebook. Within this constrained 5 GB budget we still made real progress: Qwen/Qwen3-4B in 4-bit quantization, with KV-caching enabled, thinking mode disabled, and logit-level label scoring instead of free-text JSON generation, ran reliably and was used to validate the extraction pipeline end to end on smaller batches. Once the per-slice stress tests had ruled out our own code as the cause of the memory discrepancy, we requested a dedicated, full 20 GB single-slice GPU allocation rather than continuing to work around

the shared 5 GB slices, and this request was granted. The complete 59,996-commit LLM reasoning extraction was then run on this dedicated 20 GB allocation, using the same validated Qwen3-4B configuration, without any further hardware-related constraints. We report the earlier 5 GB-slice work in detail because the throughput vs reliability trade-off it surfaced, a larger model that is individually more capable but too slow to run at corpus scale, versus a smaller model that is fast enough but needs a stricter, logit-scored output format to remain reliable, is likely to recur in any project attempting LLM-derived feature extraction at tens-of-thousands-of-commits scale on shared GPU infrastructure, even where a fuller allocation is eventually obtained.

7.6 Threats to validity and limitations

- **Compact LLM representation:** In the logit-scored seven-field representation of the LLMs, expressiveness is sacrificed for ease of extraction; using a more expressive representation (such as including the sentence embedding of the rationale generated alongside the categorical labels) could affect the results shown in Section 6.
- **Reduced/blended variants do not change the overall picture:** Experiments 7B-7F (Section 6)confirm the 7A result with respect to dimensionality reduction and ensemble combination variations: none of these improves on the 4A/4B2/4F group more than marginally, so that the qualitative result - JIT+CodeBERT as the weight-bearing pair, and the marginal contribution of the LLM stream to it being small - is not an artifact of the specific fusion method applied in 7A.
- **Label noise from SZZ:** Ultimately, all labeling relies on SZZ-style commit trace analysis, which is notorious for having a false positive rate on the order of 20% - 30% [6]. A Semantic-SZZ style re-labeling process was not performed on this dataset, so some of the errors we report may come from mislabelled commits and not from the features or models.
- **Language imbalance:** The final dataset contains 59,996 commits: approximately 36,000 Java (60%), 12,000 C++ (20%), and 11,996 Python (20%). It covers multiple projects across these languages, including Apache, CPython, Django, Flask, and Godot. Commits were chronologically split within each project into 70% training, 15% validation, and 15% testing, preserving repository evolution without random shuffling or temporal leakage. Although the results are reported overall, per-project or per-language analysis could provide further insight into generalizability.
- **Single train/test partition:** Results reflect one project-wise chronological split rather than repeated or cross-project evaluation; the rankings observed here should be treated as a single, well-controlled data point rather than a claim of generality across all JIT-SDP corpora.
- **Fixed 0.5 decision threshold:** All reported metrics other than ROC-AUC and PR-AUC use a fixed 0.5 probability threshold; threshold calibration per deployment context (e.g. optimising for a target precision to control developer alert fatigue) was not explored.

8 Conclusion and Future Work

We started off from a brief systematic survey of 37 primary JIT-SDP papers and identified three consistent gaps: lack of depth of meaning in structural features, lack of reasoning features that are explicitly stated, and virtually exclusive single-language evaluations (Table 4). We subsequently carried out an exhaustive evaluation of a hybrid JIT+CodeBERT+LLM feature fusion framework for all possible combinations of the three streams, along with two further diagnosis experiments, using our purpose-built dataset of 59,996 commits and three languages that were split in time, taken from 19 open-source projects. JIT+CodeBERT feature fusion is the single best-performing combination of features in this paper: compared with the stronger of its two baselines (JIT only), simple early fusion (4A) improves F1 by 0.0270, MCC by 0.0363, and ROC-AUC by 0.0231, and the three-model blend (4F) improves them by 0.0327, 0.0407, and 0.0248. A much-reduced 62-feature variant (4B2) matches or beats the 447-feature trimodal fusion on F1 and MCC.

That, however, does not tell the full story, and it would be a serious misreading of this work to conclude that the LLM reasoning stream does not have much value to offer. The LLM stream is valuable enough by itself; it generates a significant benefit when combined with CodeBERT, and our diagnostic tests (experiments 8 and 9, Section 6) demonstrate that its weakness within the complete 447-feature trimodal fusion was caused by redundancy in the original 51-feature representation rather than lack of meaningful signals. When the LLM representation is trimmed to 14 confidence and margin values, it remains valuable. The reduced representation keeps virtually all of the F1 and PR-AUC of the complete LLM representation, and its combination with the other two streams achieves the highest MCC observed in this work, along with higher recall than the best JIT+CodeBERT blend (4F). In other words, a much smaller and cheaper LLM feature set holds its own against every other configuration we tried. The advantage is clearest on MCC and recall, while on F1, ROC-AUC, and PR-AUC the JIT+CodeBERT blend by itself still keeps a slight edge. The results also show that our original high-dimensional LLM schema contained too much redundancy.

The next immediate steps are: firstly, producing breakdowns of all experiments by language (Java/Python/C++) to verify if the fusion benefit, as well as the LLM redundancy result, is consistently true across ecosystems or whether it is particular to the Java-centric nature of the present dataset; secondly, experimenting with a more complex representation of LLM features, e.g., keeping the sentence embedding of the produced rationale along with the labels, to determine if an even better chosen and more compact feature representation will bridge the gap on F1 and ROC-AUC as well; and thirdly, conducting a Semantic-SZZ style re-labeling phase [6] to determine how much of the residual error in all the experiments is due to label noise and not to model/feature shortcomings. Depending on the outcomes of the above experiments, the framework, dataset, and experiment configurations are ready for reuse by others, while the reproducibility instructions provided in Sections 4 and 7 are specifically meant to avoid having to re-derive from scratch the GPU budget and data mining choices made here.

Declarations

- **Funding:** Not applicable.
- **Conflict of interest/Competing interests:** The authors declare no competing interests.
- **Ethics approval and consent to participate:** Not applicable; this study analyses publicly available open-source commit metadata only.
- **Consent for publication:** Not applicable.
- **Data availability:** The mined and merged multi-language commit dataset, the extracted CodeBERT and LLM feature tables, and the experiment scripts are available in the project repository at https://github.com/PrathamKP6/Hybrid-Feature-Fusion-JIT-SDP.
- **Materials availability:** Not applicable.
- **Code availability:** All preprocessing, feature-extraction, fusion, and evaluation code is available in the repository above.
- **Author contribution:** All authors contributed equally to this work.

References

[1] Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, N.: A large-scale empirical study of just-in-time quality assurance. IEEE Trans. Softw. Eng. **39**(6), 757–773 (2013). https://doi.org/10.1109/TSE.2012.70

[2] Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., Hassan, A.E.: Studying just-in-time defect prediction using cross-project models. Empir. Softw. Eng. **21**(5), 2072–2106 (2016). https://doi.org/10.1007/s10664-015-9400-x

[3] Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: Proc. 2nd Int. Workshop on Mining Software Repositories (MSR), pp. 24–28 (2005). https://doi.org/10.1145/1083142.1083147

[4] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M.: CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1536–1547 (2020). https://doi.org/10.18653/v1/2020.findings-emnlp.139

[5] Kim, S., Shivaji, S., Whitehead, J.: A reflection on change classification in the era of large language models. IEEE Trans. Softw. Eng. **51**(3), 864–869 (2025). https://doi.org/10.1109/TSE.2025.3539566

[6] Peng, R., Xia, X., Lo, D., Li, S.: Semantic SZZ: GPT-based relabelling for bug-inducing commit identification. In: Proc. IEEE/ACM 47th Int. Conf. Software Engineering (ICSE) (2025).

[7] Haldar, S., Capretz, L.F.: Feature importance in the context of traditional and just-in-time software defect prediction models. In: Proc. IEEE Canadian

Conf. Electrical and Computer Engineering (CCECE), pp. 818–822 (2024). https://doi.org/10.1109/CCECE59415.2024.10667167

[8] Kumar, A., Maram, B., Chandre, S.: Enhancing software defect prediction accuracy through feature selection-driven hybrid intelligence. In: Proc. Third Int. Conf. Industry 4.0 Technology (I4Tech), pp. 1–5. IEEE (2025). https://doi.org/10.1109/I4Tech64670.2025.11277790

[9] Chen, L.-Q., Wang, C., Song, S.-L.: Software defect prediction based on nested-stacking and heterogeneous feature selection. Complex Intell. Syst. **8**, 3333–3348 (2022). https://doi.org/10.1007/s40747-022-00676-y

[10] Khleel, N.A.A., Nehéz, K., Fadulalla, M., Hisaen, A., et al.: Ensemble-based machine learning algorithms combined with near miss method for software bug prediction. Int. J. Netw. Distrib. Comput. **13**, 11 (2025). https://doi.org/10.1007/s44227-024-00044-x

[11] Hoang, T., Dam, H.K., Kamei, Y., Lo, D., Ubayashi, N.: DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction. In: Proc. IEEE/ACM 16th Int. Conf. Mining Software Repositories (MSR), pp. 34–45 (2019). https://doi.org/10.1109/MSR.2019.00016

[12] Zhou, X., Han, D., Lo, D.: Bridging expert knowledge with deep learning techniques for just-in-time defect prediction. Empir. Softw. Eng. **30**(1), 37 (2025). https://doi.org/10.1007/s10664-024-10591-0

[13] Yadav, M., Kumar, L., Passricha, V.: A comprehensive approach to enhance fault prediction through code-comment analysis with CodeBERT. Appl. Intell. **55**(15), 1010 (2025). https://doi.org/10.1007/s10489-025-06820-8

[14] Abu Talib, M., Bou Nassif, A., Azzeh, M., Alesh, Y., Afadar, Y.: Parameter-efficient fine-tuning of pre-trained code models for just-in-time defect prediction. Neural Comput. Appl. **36**(27), 16911–16940 (2024). https://doi.org/10.1007/s00521-024-09930-5

[15] Chen, Y., Huang, Q., Zhou, Y., Li, Z.: JIT-Align: Semantic alignment for CodeT5-based just-in-time defect prediction. Inf. Softw. Technol. **172**, 107492 (2025).

[16] Zhao, Y., Damevski, K., Chen, H.: A systematic survey of just-in-time software defect prediction. ACM Comput. Surv. **55**(10), 201:1–201:35 (2023). https://doi.org/10.1145/3567550

[17] Keshavarz, H., Nagappan, M.: ApacheJIT: A large dataset for just-in-time defect prediction. In: Proc. 19th IEEE/ACM Int. Conf. Mining Software Repositories (MSR), pp. 191–195 (2022). https://doi.org/10.1145/3524842.3527996

[18] Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), pp. 785–794 (2016). https://doi.org/10.1145/2939672.2939785

[19] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y.: LightGBM: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 3146–3154 (2017).

[20] Spadini, D., Aniche, M., Bacchelli, A.: PyDriller: Python framework for mining software repositories. In: Proc. 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 908–911 (2018). https://doi.org/10.1145/3236024.3264598

[21] Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Yang, M., Wang, P., Zhu, Q., Men, R., Luo, S., Li, T., Tang, T., Yin, W., Yu, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Wan, Y., Liu, Y., Cui, Z., Zhou, Z., Qiu, Z., et al.: Qwen3 Technical Report. arXiv preprint arXiv:2505.09388 (2025). https://doi.org/10.48550/arXiv.2505.09388

[22] Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: QLoRA: Efficient finetuning of quantized LLMs. In: Advances in Neural Information Processing Systems (NeurIPS), **36**, 10088–10115 (2023).

[23] Wang, K., Li, S., Xia, X., Lo, D.: An empirical study of just-in-time defect prediction using various machine learning techniques. Int. J. Comput. Inf. Syst. Ind. Manag. Appl. **16**, 1–18 (2024).

[24] Tan, L., Wu, H., Zhang, Y.: CatBoost with entropy and file-age features for cross-repository defect prediction. J. Syst. Softw. **212**, 112024 (2025).

[25] Chowdhury, S., Uddin, G., Hemmati, H., Holmes, R.: Method-level bug prediction: Problems and promises. ACM Trans. Softw. Eng. Methodol. **33**(4), 1–34 (2024). https://doi.org/10.1145/3640331

[26] Zhan, W., Mo, R., Jiang, Y., Wang, D.: Just-in-time prediction of software architectural changes through commit-level analyses. IEEE Trans. Softw. Eng. **51**(8), 2285–2304 (2025). https://doi.org/10.1109/TSE.2025.3587849

[27] Chauhan, V., Arora, C., Khalajzadeh, H., Grundy, J.: Managing human-centric software defects: Insights from GitHub and practitioners' perspectives. arXiv preprint arXiv:2408.01621 (2024). https://doi.org/10.48550/arXiv.2408.01621

[28] Pujar, S., Zheng, Y., Buratti, L., Lewis, B., Chen, Y., Laredo, J., Morari, A., Epstein, E., Lin, T., Yang, B., Su, Z.: Analyzing source code vulnerabilities in the D2A dataset with ML ensembles and C-BERT. Empir. Softw. Eng. **29**, 48 (2024). https://doi.org/10.1007/s10664-023-10405-9

[29] Aleithan, R.: Explainable just-in-time bug prediction: Are we there yet? In: Proc. IEEE/ACM 43rd Int. Conf. Software Engineering: Companion Proceedings (ICSE-Companion), pp. 129–131 (2021). https://doi.org/10.1109/ICSE-Companion52605.2021.00056

[30] Harzevili, N.S., Belle, A.B., Wang, J., Wang, S., Jiang, Z.M., Nagappan, N.: A systematic literature review on automated software vulnerability detection using machine learning. ACM Comput. Surv. **57**(3), 55:1–55:36 (2025). https://doi.org/10.1145/3699711

[31] Zhang, J., Li, Y., Bai, Y., Zhao, Y.: TSBoostDF: Transfer boosting with similarity thresholds for cross-project defect prediction. Inf. Softw. Technol. **148**, 106929 (2022).

[32] Feyzi, F., Daneshdoost, A.: Studying the effectiveness of deep active learning in software defect prediction. Int. J. Comput. Appl. **45**(7–8), 534–552 (2023). https://doi.org/10.1080/1206212X.2023.2252117

[33] Cabral, G.G., Minku, L.L., Oliveira, A.L.I., Pessoa, D.A., Tabassum, S.: An investigation of online and offline learning models for online just-in-time software defect prediction. Empir. Softw. Eng. **28**, 121 (2023). https://doi.org/10.1007/s10664-023-10335-6

[34] Wu, Q., Wang, X., Wei, D., Chen, B., Dang, Q.: Just-in-time software defect prediction method for non-stationary and imbalanced data streams. Softw. Qual. J. **33**(1), 14 (2025). https://doi.org/10.1007/s11219-025-09711-w

[35] Yamamoto, H., Wang, D., Rajbahadur, G.K., Kondo, M., Kamei, Y., Ubayashi, N.: Towards privacy preserving cross project defect prediction with federated learning. In: IEEE Int. Conf. on Software Analysis, Evolution and Reengineering (SANER), pp. 485–496 (2023). https://doi.org/10.1109/SANER56733.2023.00052

[36] Li, Z., Du, Q., Zhang, H., Jing, X.-Y., Wu, F.: An empirical study of data sampling techniques for just-in-time software defect prediction. Autom. Softw. Eng. **31**(2), 56 (2024). https://doi.org/10.1007/s10515-024-00455-8