Commit Messages in Context: LLM-Based Semantic Relabeling Improves Mobile Crash Prediction

Abstract

Accurately predicting crashing releases in mobile applications depends on the quality of ground-truth labels derived from software repositories. Prior approaches have relied on heuristic keyword matching within commit logs, an approach that lacks semantic understanding and introduces substantial label noise. As a result, existing crash-prediction models exhibit limited reliability and inconsistent generalization. In this paper, we present a Large Language Model (LLM)-based semantic relabeling framework that redefines how crash-related commits are identified. Unlike traditional keyword-based heuristics, our method interprets the contextual meaning of developer messages to distinguish genuine crash fixes from unrelated technical changes. The resulting semantically enriched labels form the basis of a newly released dataset, which we make publicly available to support reproducible research in mobile software analytics. Using this refined dataset, we quantify the downstream impact of label quality and benchmark both conventional and extended machine learning models, including *Random Forest*, *Gradient Boosting*, *XGBoost*, and *LightGBM*. Our preliminary experimental results across ten open-source Android projects demonstrate consistent improvements, with F1-score gains up to +0.19 and AUC reaching 0.82. Together, these findings establish semantic relabeling and open-data benchmarking as a robust foundation for advancing crash-release prediction research.

ACM Reference Format:

. 2026. Commit Messages in Context: LLM-Based Semantic Relabeling Improves Mobile Crash Prediction. In . ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Mobile application stability is a critical determinant of commercial success. Application crashes directly impact user satisfaction, retention rates, and revenue [11, 14]. Users quickly abandon unstable applications and express dissatisfaction through negative reviews, which discourages potential new users. Consequently, the ability to predict *crashing releases* is an important topic.

Foundational work by Xia et al. [14] demonstrated the feasibility of crash release prediction by training classifiers on repository metrics and commit logs. However, their reported performance was not sufficient for practical adoption. A key limitation lies in their reliance on fixed-keyword labeling. Specifically, releases were labeled as "crashing" if the subsequent release contained a commit message with predefined keywords such as "crash." While simple, this approach lacks semantic understanding; it cannot distinguish between phrases like "fixed a crash" and "refactored the crash reporting library." As a result, it produces noisy ground-truth labels, leading to both false positives and false negatives and therefore constraining the model performance.

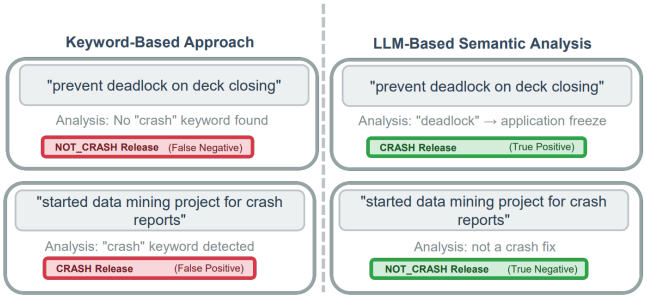

Figure 1: Comparison of labeling approaches. Keyword-based matching (left) produces false negatives and false positives, while LLM-based semantic analysis (right) correctly classifies those commits through contextual understanding.

To address this limitation, we propose a new approach that leverages **large language models (LLMs)** to perform semantic relabeling of commit messages. Instead of keyword matching, our method interprets the intent and contextual meaning of developer comments to identify genuine crash-related fixes. By generating higher-quality labels, we enable state-of-the-art machine learning models to better capture predictive patterns associated with crashing releases.

Figure 1 illustrates these limitations with concrete examples from the dataset [14]. The first example shows a commit message stating "prevent deadlock on deck closing" which clearly addresses a crash-related issue, as deadlocks cause applications to freeze or become unresponsive. However, the keyword-based approach fails to identify this commit because it lacks the semantic understanding to recognize that technical terms like "deadlock" directly lead to user-facing crashes. Conversely, the second example demonstrates the opposite problem: a commit message "started data mining project for crash reports" is incorrectly labeled as crash-related by the keyword-based approach simply because it contains the word "crash", despite clearly describing the initiation of an analytics project rather than fixing an actual crash. The LLM-based approach correctly identifies both cases through a semantic understanding of the commit context.

Our contributions are as follows: (1) **Semantic labeling method:** We introduce a contextual labeling approach that replaces fixed-keyword matching with LLM-based analysis of commit messages to improve ground-truth reliability; and (2) **Open dataset**[1] We

[1]Dataset: https://anonymous.4open.science/r/Crash_Realease_Data-E059/README.md

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

release the newly relabeled dataset to support further research in crash-release prediction and software analytics; and (3) **Empirical benchmarking:** We quantify the impact of improved label quality through benchmark experiments comparing both conventional and extended machine learning models, demonstrating F1-score gains of up to +0.19 and AUC improvements to 0.82. Together, these contributions provide early evidence that semantic relabeling is a promising direction for advancing crash release prediction research.

2 Related Work

Predicting Mobile Crashes. Xia et al.[14] first framed crash-release prediction for mobile applications using repository metrics and commit logs. However, their approach relies on fixed-keyword matching to label releases, which lacks semantic understanding and produces noisy labels. Our work differs fundamentally as we replace keyword matching with LLM-based semantic analysis to distinguish genuine crash fixes from unrelated mentions, thereby improving label quality and predicting model performance.

Predicting Software Defects. Research in the closely related field of Just-In-Time (JIT) defect prediction, which focuses on identifying risky commits rather than releases, has extensively demonstrated the value of commit log data. Kamei et al. [9] established commit-level defect prediction using repository-based features, however, their approach lacks a semantic understanding of commit content. Hoang et al. [8] developed DeepJIT, an end-to-end deep learning approach for commit-level defect prediction. Their framework leverages CNNs to automatically extract semantic features from commit messages and code changes, demonstrating advanced performance compared to traditional metric-based models.

A recent systematic literature review by Giray et al. [7] on the use of deep learning for software defect prediction provides a broader context. Their findings, covering 102 studies, indicate that while advanced models are increasingly applied, many approaches still rely on manually engineered features that often do not rely on the semantic information of code changes.

This supports the premise that a more semantic-aware analysis of developer-generated text is a promising direction for improving model accuracy. This work aims to address this gap by adapting the principles of semantic commit analysis to the challenge of labeling and predicting entire crashing releases.

LLM for Labeling. Ahmed et al.[1] investigates using LLMs to replace human annotators in software engineering tasks, examining ten annotation tasks, including code summarization evaluation and static analysis warning assessment. They find that LLM-human agreement can match human-human agreement for certain tasks, with model-model agreement serving as a predictor of suitability. Tan et al.[13] establishes LLM-based annotation as a promising alternative to manual labeling, though challenges such as hallucinations and bias remain. Chen et al. [3] employs LLMs as annotators by including training samples in prompts to ensure generated labels align with dataset distributions, successfully addressing data scarcity. We build on this foundation by applying LLM-based semantic labeling to crash-release prediction, addressing the label noise caused by keyword-based approaches.

Table 1: Statistics for Android projects. #R: total releases, #C: crashing releases, %C: percentage of crashing releases, ΔC: label changes from previous dataset[14], FP: false positives, FN: false negatives

Project	Period	#R	#C	%C	ΔC	FP	FN
ankidroid Anki-Android	09/06–14/09	591	109	18	50	32	18
bpellin keepassdroid	09/01–13/09	149	25	17	10	6	4
BrandroidTools OpenExplorer	11/06–13/05	184	40	22	20	16	4
freezy android-xbmcremote	09/08–13/12	391	33	8	24	19	5
gothfox Tiny-Tiny-RSS	11/09–14/07	280	36	13	10	5	5
guardianproject Gibberbot	10/03–14/09	223	27	12	8	5	3
mariotaku twidere	12/04–14/05	262	35	13	5	3	2
mtotschnig MyExpenses	11/03–14/09	241	34	14	8	4	4
qii weiciyuan	12/07–14/07	204	36	18	11	4	7
WSDOT wsdot-android-app	10/05–14/07	122	9	7	1	1	0
Total	–	**2647**	**384**	**15**	**140**	**95**	**52**

3 Dataset

Our study is based on the dataset used by Xia et al. [14]. The dataset comprises ten open-source Android repositories, each annotated with twenty features derived from repository statistics. In addition, the dataset includes commit messages, which serve as the basis for both the original crash-related labeling and our new semantic relabeling approach.

Table 1 provides a summary of the dataset used in this study. It contains ten open-source Android application repositories covering the period from 2009 to 2014. The number of releases per project ranges from 122 to 591, while the proportion of crashing releases varies between 7% and 22%. The difference in total crash labels **ΔC** is **140**.

To ensure comparability with prior work, we adopted the same repository metrics used by Xia et al. [14]. These features capture five key dimensions: (1) **Complexity** metrics (e.g., cyclomatic complexity), (2) **Time** metrics (e.g., days since previous release), (3) **Code** metrics (e.g., lines added/deleted, file counts), (4) **Diffusion** metrics (e.g., file entropy), and (5) **Commit** and **Text** metrics (e.g., fuzzy scores from commit logs). For a complete description of all metrics, their definitions, and rationale, can be found in the original work by Xia et al. [14]. This approach maintains feature consistency while enabling direct performance comparison with the baseline study.

4 Methodology

4.1 Problem Formulation

Let $R = \{r_1, r_2, \ldots, r_n\}$ denote mobile app releases with commit messages $M(r_i) = \{c_1, c_2, \ldots, c_k\}$ from the next release r_{i+1}.

We define a binary label $y : R \rightarrow \{0, 1\}$ where $y(r_i) = 1$ indicates r_i introduced user-facing crashes. Existing approaches generate labels via keyword matching on $M(r_i)$.

Objective: Generate refined labels $\hat{y}(r_i)$ that reduce label noise by incorporating **semantic context**, leading to retrained models with improved predictive performance, formally expressed as:

$$F1Score_{\text{new}} > F1Score_{\text{baseline}}.$$

4.2 Semantic Relabeling

To achieve semantic labeling for each release, we employed deepseek-chat[2] to analyze commit messages from subsequent releases. Unlike keyword-based matching, our approach leverages the model's semantic understanding to distinguish between crash-related fixes and unrelated technical changes.

4.2.1 Prompt Design. The LLM is positioned as a software QA analyst performing a **binary classification task**. Given the commit messages $M(r_i)$ for release r_i, the model predicts:

$$\hat{y}(r_i) \in \{\texttt{CRASH}, \texttt{NOT\_CRASH}\}.$$

We define a set of **crash-indicating terms**:

$$\mathcal{T}_{\text{crash}} \supseteq \{\textit{crash}, \textit{freeze}, \textit{hangs}, \textit{unresponsive}, \textit{force close}\}, \quad (1)$$

and a set of purely **technical terms** that should *not* trigger a positive label:

$$\mathcal{T}_{\text{technical}} \supseteq \{\texttt{exception}, \texttt{memory leak}, \texttt{error}, \texttt{bug}\}. \quad (2)$$

Exclusion Rule. A critical component of our prompt design is the explicit exclusion rule. We formalize that for any term $t \in \mathcal{T}_{\text{technical}}$, we require:

$$\forall t \in \mathcal{T}_{\text{technical}} : (t \in M(r_i)) \nRightarrow (\hat{y}(r_i) = \texttt{CRASH}). \quad (3)$$

This formulation filters purely technical discussions (e.g., "refactored exception handler") that lack user-facing crash implications, **reducing false positives**.

The model is further required to provide textual evidence supporting each classification, either a direct quote from commit messages or "None" if no evidence is found. This step improves interpretability and enables manual verification.

We used few-shot prompting [2], supplying five representative examples of both positive (crash-related) and negative (technical-only) cases.

Figure 2 provides an overview of our complete methodology workflow, covering semantic labeling, model training, and performance evaluation.

4.2.2 Labeling Process. For each release r_i in our dataset, we extracted all commit messages $\{c_1, c_2, ..., c_n\}$ from the subsequent release r_{i+1}. These messages were concatenated and submitted to deepseek-chat via API as a single query along with the structured prompt. We set the temperature parameter to 0.1 to ensure deterministic and reproducible outputs across multiple runs. The model's binary classification (CRASH or NOT_CRASH) and supporting evidence were stored for each release.

4.3 Model Benchmarking

Based on the newly generated labels, we retrain the same baseline models as used in the prior work [14], namely: **k**-Nearest Neighbors (KNN), Naïve Bayes, Decision Tree, and Random Forest.
All models use the same feature set as in the original study, with the only difference being the use of our semantically refined labels. Model performance is evaluated using 10-fold stratified cross-validation, following the same protocol as in prior work.
This approach ensures more stable performance estimates while maintaining class distribution across folds, which is important given the inherent imbalance between crashing and non-crashing releases.

[2]deepSeek-chat accessed via API at https://api-docs.deepseek.com/ in [October 2025].

4.4 Extended Model Benchmarking

To benchmark the effectiveness of the dataset with machine learning approaches on our semantically relabeled dataset, we trained and evaluated five state-of-the-art models that have demonstrated strong performance in classification tasks across various domains:

- **XGBoost** [4]: An optimized gradient boosting implementation known for its computational efficiency and regularization capabilities, which help prevent overfitting.
- **LightGBM** [10]: A gradient boosting framework that uses tree-based learning algorithms, designed for distributed and efficient training with large datasets.
- **Gradient Boosting (GBM)** [6]: A traditional gradient boosting machine that builds an ensemble of weak learners sequentially, with each tree correcting the errors of its predecessors.
- **AdaBoost** [5]: An adaptive boosting algorithm that adjusts the weights of misclassified instances, forcing subsequent learners to focus on harder cases.
- **Multi-Layer Perceptron (MLP)** [12]: A feedforward neural network with multiple hidden layers, capable of learning non-linear decision boundaries through backpropagation.

All models were trained using the same feature set as the baseline approaches, maintaining consistency in our experimental design.

5 Results

To benchmark the effectiveness of our LLM-based semantic relabeling approach, we compare the performance of baseline models trained on both the original keyword-based labels and our semantically relabeled labels. Additionally, we benchmark a set of extended models trained exclusively on the new, semantically relabeled dataset.

Table 2 summarizes the performance comparison between the original keyword-based labeling [14] and our LLM-based semantic relabeling. All models show a consistent improvement in F1-score ΔF1, with the Random Forest achieving the largest gain of +0.19.

Table 3 presents the results of training the extended models. A 10-fold stratified cross-validation was used to train all models. The best performing models were LightGBM and AdaBoost, both achieving an F1-score of 0.45. Based on the AUC metric, LightGBM is the best performing model with a score of 0.82.

5.1 Label Validation

To assess the quality of our LLM-based relabeling approach, we conducted a validation analysis. The majority of releases (94.45%) retained their original labels, suggesting general agreement between keyword-based and semantic approaches. For releases where labels changed, we performed a manual inspection of a random sample of 40 cases. Our analysis confirmed that the new labels were reasonable, with changed label primarily justified by the presence of crash-indicating language in commit messages that the keyword-based approach had missed or misclassified due to a lack of contextual understanding. This validation supports the reliability of our semantic labeling methodology.

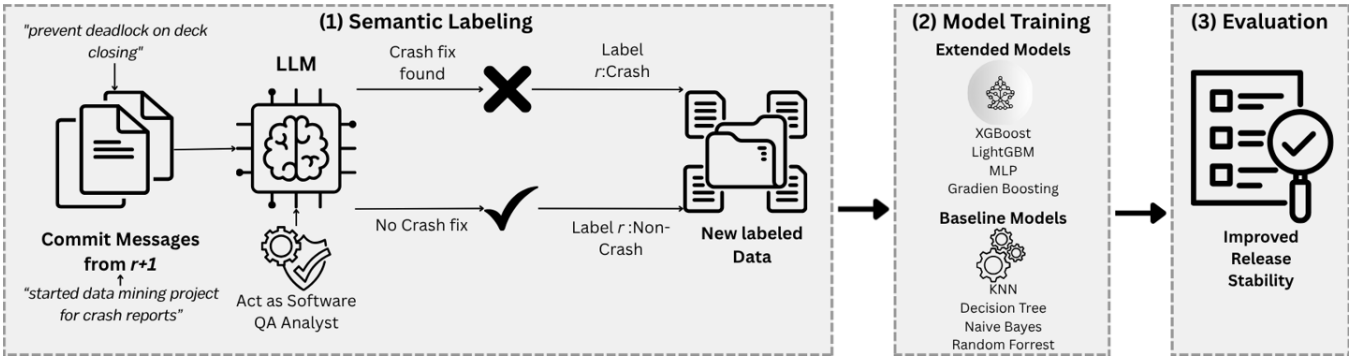

Figure 2: Overview of the proposed methodology. Commit messages are semantically analyzed by an LLM to generate crash labels (Stage 1), these are then used to train baseline and extended machine learning models (Stage 2), which increase the Crash Predicting performance(Stage 3).

Table 2: Original keyword labels vs. LLM relabeling per model. Δ = LLM − Original.

	Precision			Recall			F1			AUC		
Model	Orig	LLM	Δ	Orig	LLM	Δ	Orig	LLM	Δ	Orig	LLM	Δ
Naïve Bayes	0.20	**0.21**	+0.01	0.62	**0.74**	+0.12	0.30	**0.31**	+0.01	0.64	**0.69**	+0.05
kNN	0.17	**0.25**	+0.08	0.60	**0.67**	+0.07	0.26	**0.37**	+0.11	0.60	**0.70**	+0.10
Decision Tree (C4.5)	0.21	**0.27**	+0.06	0.39	**0.68**	+0.29	0.27	**0.38**	+0.11	0.58	**0.68**	+0.10
Random Forest	0.23	**0.32**	+0.09	0.30	**0.75**	+0.45	0.26	**0.45**	+0.19	0.63	**0.81**	+0.18

Table 3: Performance of extended models on the LLM-relabeled dataset. Values show metric and (improvement over the best original baseline). Best original baselines[14]: Precision=0.23, Recall=0.62, F1=0.30, AUC=0.64.

Model	Precision	Recall	F1	AUC
GBM	0.31 (+0.08)	**0.76** (+0.14)	0.44 (+0.14)	0.81 (+0.17)
MLP	0.31 (+0.08)	0.71 (+0.09)	0.42 (+0.12)	0.75 (+0.11)
LightGBM	0.32 (+0.09)	**0.76** (+0.14)	**0.45** (+0.15)	**0.82** (+0.18)
XGBoost	0.31 (+0.08)	**0.76** (+0.14)	0.44 (+0.14)	0.81 (+0.17)
AdaBoost	**0.33** (+0.10)	0.71 (+0.09)	**0.45** (+0.15)	0.80 (+0.16)

5.2 Error Analysis

A qualitative inspection of misclassified cases revealed two dominant error sources in keyword-based labeling: (1) false negatives caused by implicit crash indicators such as deadlocks, ANR states, and freezes, and (2) false positives triggered by non-crash mentions like "crash reporting library" or test-case descriptions. The LLM correctly identified these semantic distinctions, explaining why the refined labels improved model performance across all classifiers.

6 Threats To Validity

Our preliminary investigation has several constraints. We focus on commit message analysis using a single LLM (DeepSeek) to establish initial feasibility. Some labels may be affected by LLM hallucinations [1]. The evaluation uses the established Xia et al. [14] dataset (10 open-source Android apps) to enable direct comparison with prior work. While this demonstrates that semantic relabeling improves upon keyword-based approaches, generalization requires further validation.

7 Conclusion and Future Work

In this paper, we present an LLM-based semantic relabeling framework that addresses the fundamental limitations of keyword-based crash labeling in mobile application repositories. Unlike approaches that lack contextual understanding, our method leverages large language models to interpret the semantic meaning of commit messages, distinguishing genuine crash fixes from unrelated technical changes. Experimental results across ten open-source Android projects demonstrate consistent improvements across all models, with F1-score gains of up to +0.19 and AUC reaching 0.82. We release the dataset to support reproducible research. Together, these contributions provide early evidence that semantic relabeling is a promising direction for advancing mobile crash release prediction.

Future work will focus on four main directions. First, we plan to validate our approach with other prominent LLMs, including Claude 4.5, GPT-5, and Gemini 2.5 Pro. Second, we aim to incorporate more recent data to increase the dataset size and capture evolving patterns in mobile app development and failure characteristics. Third, we will expand our dataset to include a wider variety of repositories, enabling more generalized findings across mobile apps. Additionally, a comprehensive manual validation of a larger subset of the relabeled releases will be conducted to establish inter-rater reliability metrics and further quantify the improvement in label quality.

References

[1] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. 2025. Can llms replace manual annotation of software engineering artifacts?. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 526–538.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Ruirui Chen, Chengwei Qin, Weifeng Jiang, and Dongkyu Choi. 2024. Is a large language model a good annotator for event extraction?. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 38. 17772–17780.

[4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[5] Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139. doi:10.1006/jcss.1997.1504

[6] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[7] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, and Bedir Tekinerdogan. 2023. On the use of deep learning in software defect prediction. *The Journal of Systems and Software* 195 (2023), 111537.

[8] Thong Hoang, Hoa Khanh Dam, Yasutaka Kamei, David Lo, and Naoyasu Ubayashi. 2019. DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 34–45.

[9] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering* 39, 6 (2013), 757–773.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[11] Unnati Narang, Venkatesh Shankar, and Sridhar Narayanan. 2018. *The impact of mobile app failures on online and offline purchases.* Technical Report.

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation.* Technical Report.

[13] Zhen Tan, Dawei Li, Song Wang, Alimohammad Beigi, Bohan Jiang, Amrita Bhattacharjee, Mansooreh Karami, Jundong Li, Lu Cheng, and Huan Liu. 2024. Large language models for data annotation and synthesis: A survey. *arXiv preprint arXiv:2402.13446* (2024).

[14] Xin Xia, Emad Shihab, Yasutaka Kamei, David Lo, and Xinyu Wang. 2016. Predicting crashing releases of mobile applications. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.