VFCionX: Bridging Large and Small Models for Robust Vulnerability-Fixing Commit Identification

Xing Cui[1,2], Jingzheng Wu[1*], Wenxiang Ou[1], Tianyue Luo[1], Zhiyuan Li[1,2], Xiang Ling[1,3]

[1] Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] University of Chinese Academy of Sciences, Beijing 100190, China
[3] Quan Cheng Laboratory, Jinan 250103, Shandong, China

{cuixing, jingzheng08, ouwenxiang, tianyue, lingxiang}@iscas.ac.cn, lizhiyuan221@mails.ucas.ac.cn

Abstract

Vulnerability-Fixing Commit Identification(VFCI) is a critical task in software security maintenance that aims to automatically identify code commits that patch security vulnerabilities. However, existing approaches face challenges in handling low-quality commit messages and entangled commits, which limit their identification performance. To address these issues, we propose VFCionX, a novel VFCI framework that integrates large and small language models in a collaborative architecture. VFCionX consists of three core modules: Message Classifier, Patch Classifier, and Ensemble Classifier. The Message Classifier employs a multi-source contextual augmentation strategy to enhance the quality of commit messages and fine-tunes the Qwen2.5-1.5B model, significantly improving classification performance in the textual modality. The Patch Classifier combines heuristic rules with a Qwen2.5-Coder-7B-driven file selector to filter noise from entangled commits, and incorporates a line-level feature extractor based on CodeBERT and CNN to capture local pattern differences between added and deleted code lines. The Ensemble Classifier integrates predictions from both channels using the AdaBoost algorithm, enhancing model robustness and generalization. Experimental results on five popular C/C++ repositories comprising 24,630 commits show that VFCionX achieves an F1-score of 81.47%, outperforming the best baseline by 9.42%. Ablation studies validate the effectiveness of each component, while sensitivity analysis reveals optimal parameter settings for balancing performance and noise resilience. This work provides a new and effective solution for robust vulnerability patch identification.

1 Introduction

As modern software development continues to evolve toward modular and component-based architectures, software systems increasingly depend on open-source software (OSS), which introduces substantial security risks due to hidden vulnerabilities (Ladisa et al. 2023; Boughton et al. 2024; Liu et al. 2022). In practical settings, monitoring public vulnerability databases such as the National Vulnerability Database (NVD) and CVE Details provides timely access to newly disclosed vulnerabilities and corresponding patches, thereby supporting effective vulnerability management. However, OSS vulnerability disclosure often follows Coordinated Vulnerability Disclosure (CVD) practices (Ponta, Plate, and Sabetta 2020), where security researchers privately report vulnerabilities to vendors and allow time for remediation before public disclosure. This process results in a temporal gap between patch submission and the official disclosure of the vulnerability (Li and Paxson 2017; Sabetta and Bezzi 2018). During this window, attackers may exploit undisclosed vulnerabilities, increasing the risk of system compromise. Therefore, automatically identifying vulnerability-fixing commits before public disclosure and extracting latent vulnerability information is critical for enabling proactive defense and facilitating early patching of affected OSS components.

Vulnerability-Fixing Commit Identification (VFCI) is a challenging task. Due to limited resources and lack of security expertise, developers often find it difficult to manually analyze all commits to detect vulnerability fixes. Therefore, developing automated tools for VFCI is of significant importance. Several tools have been proposed to address this task. Zhou et al. introduce VulFixMiner, which fine-tunes CodeBERT to encode code changes into embedding vectors and employs a single-layer neural network for classification (Zhou et al. 2021a). Nguyen et al. enhance VulFixMiner by incorporating additional data sources such as commit messages and issue reports, resulting in the VulCurator framework (Nguyen et al. 2022). Nguyen et al. also propose MiDas, a multi-granularity VFCI framework that decomposes code changes into different levels, extracts features for each level, and uses an ensemble model to generate final predictions (Nguyen et al. 2023). The advanced understanding capabilities of large language models (LLMs) in both natural and programming languages offer new opportunities for VFCI. Recent studies (Li et al. 2024; Ding et al. 2024) leverage LLMs such as GPT-4 (Achiam et al. 2023) with prompt learning and chain-of-thought (CoT) reasoning (Lyu et al. 2023) to identify fixing commits, while others adopt fine-tuned open source models such as CodeLlama-7B (Roziere et al. 2023) to achieve similar goals.

Although VFCI has achieved notable progress in recent years, several key challenges remain. (1) Limited utilization of commit messages. Although CVD recommends omitting

*Corresponding author

vulnerability-related information from commit messages, this guideline is not followed consistently. Many messages still contain contextual clues suggesting vulnerability fixes, resulting in uncertainty regarding their effective use. For example, in the Linux kernel project, "commit 9805187" explicitly states in its message that it fixes a potential crash or undefined behavior caused by a null pointer dereference (Github 2025). (2) Low and inconsistent message quality. Most existing methods focus on code-level features and lack systematic modeling of commit messages as a complementary signal (Zhou et al. 2021a; Nguyen et al. 2023; Ding et al. 2024). Although some approaches incorporate message content (Nguyen et al. 2022; Li et al. 2024), they overlook the quality variation between messages. Many are vague or semantically sparse, such as "fixed #2563" in CVE-2023-4683, providing limited value for reliable inference (Wu et al. 2025). (3) Entangled and noisy commit changes. Commits frequently include heterogeneous modifications, only some of which are related to vulnerability fixing. Studies report that approximately 80% of unrelated edits involve tests, general bug fixes, feature enhancements, refactoring, or documentation updates (Li et al. 2024). Most methods treat all changes equally, lacking fine-grained filtering, which introduces noise and impairs identification performance.

To address the above challenges, we propose VFCionX, a novel VFCI approach that integrates LLMs and small language models (SLMs) in a collaborative framework. VFCionX consists of three core modules: Message Classifier, Patch Classifier, and Ensemble Classifier. The message classifier improves the quality of the commit message by incorporating associated issues and pull requests. It then fine-tunes the Qwen2.5-1.5B model (Hui et al. 2024) using the enriched commit data through supervised fine-tuning (SFT) (Dong et al. 2023; Luo et al. 2023) to perform commit-level classification. The Patch Classifier employs heuristic rules and a Qwen2.5-Coder-7B-based file selector (Qwen 2025) to filter out unrelated code in entangled commits, identifying the files most relevant to the commit message. It extracts added and deleted lines, encodes them using CodeBERT (Feng et al. 2020) and a convolutional neural network (CNN) (Kim 2014), and performs classification based on fused fine-grained features. Finally, the Ensemble Classifier adopts an Adaboost-based ensemble strategy (Ying et al. 2013) to combine the predictions of both classifiers, improving generalization and mitigating the effects of noise and data imbalance.

Our contributions are summarized as follows:
(1) We propose VFCionX, a novel hybrid architecture that combines the strengths of LLM (Qwen2.5 series) and small SLM (CodeBERT and CNN) through an Adaboost-based ensemble strategy, significantly improving the accuracy and robustness of VFCI.
(2) We introduce a commit message augmentation method and a file selection mechanism to address the challenges of underutilized messages and entangled commits.
(3) We conduct extensive experiments to evaluate the effectiveness of VFCionX. On a real-world C/C++ dataset, VFCionX achieves an F1-score of 81.47%, outperforming the best baseline by 9.42%. Ablation studies further confirm the effectiveness of each component.

2 Related Work

In the field of VFCI, early studies primarily adopt traditional machine learning methods (Perl et al. 2015; Sabetta and Bezzi 2018; Zhou and Sharma 2017), which rely on handcrafted features and exhibit limited generalization. Subsequent work explores approaches based on deep learning. Zhou et al. propose VulFixMiner (Zhou et al. 2021a), which fine-tunes CodeBERT (Feng et al. 2020) to embed code changes and employs a single-layer neural network for classification. Nguyen et al. extend this work with VulCurator (Nguyen et al. 2022), which uses RoBERTa (Liu et al. 2019) to encode commit messages and issue reports, and CodeBERT for code changes, followed by logistic regression to generate predictions. Later, Nguyen et al. propose MiDas (Nguyen et al. 2023), a multi-granularity framework that trains separate base models for different levels of code change granularity and combines them via an ensemble model. While these deep learning methods improve accuracy and generalization, they have limitations. Most fail to handle entangled commits, treating all modified files equally, and some do not leverage commit messages at all. Recently, the rapid advancement of LLM (Kojima et al. 2022; Wu et al. 2023) offers new opportunities for VFCI, due to their strong understanding of both natural and programming languages. Researchers have explored LLM-based methods using GPT-4 (Achiam et al. 2023), prompt learning (Giray 2023; White et al. 2023), and chain-of-thought (CoT) reasoning (Lyu et al. 2023) for VFCI tasks, as well as fine-tuning open source models such as CodeLlama-7B (Meta 2025a). However, experimental results show that LLM-based approaches often underperform SLM like CodeBERT in vulnerability fix identification, indicating that LLMs still struggle to distinguish subtle vulnerability fixes from benign changes.

3 Problem Definition

This study addresses the task of VFCI, which aims to determine whether a given commit in a software repository corresponds to a vulnerability fix. Formally, let C_i denote a commit uniquely identified by index i. Each commit consists of two components: a commit message X_i^m, which provides a textual description of the change, and code modifications X_i^c, which include the lines of code added and deleted. The objective is to learn a binary classification function f that takes (X_i^m, X_i^c) as input and predicts a label $y_i \in \{0, 1\}$, where $y_i = 1$ indicates a vulnerability-fixing commit and $y_i = 0$ denotes a non-vulnerability-fixing commit. The goal is to identify a mapping function

$$f : (X_i^m, X_i^c) \rightarrow y_i$$

that maximizes prediction accuracy on unseen commits.

4 Methodology

This section presents the proposed method, VFCionX, which performs VFCI both efficiently and accurately. VFCionX adopts a hybrid architecture that combines LLM and

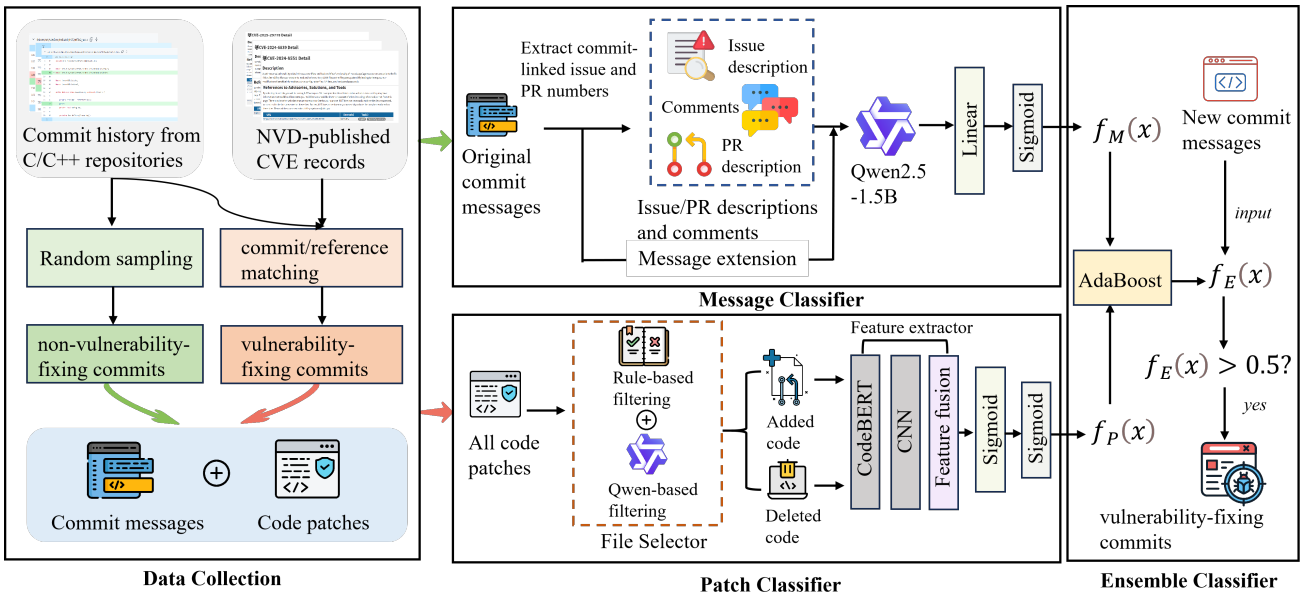

Figure 1: Overview of the VFCionX framework. The framework constructs a labeled dataset by integrating commit history from C/C++ repositories with NVD-published CVE records. It jointly analyzes commit messages and code patches using a dual-channel architecture: the message classifier enhances textual features via contextual augmentation, while the patch classifier captures structural changes through a CodeBERT-CNN pipeline. An AdaBoost-based ensemble module combines both predictions to produce the final classification result.

SLM to balance performance and computational efficiency. It consists of three key modules: the Message Classifier, the Patch Classifier, and the Ensemble Classifier. The overall framework of VFCionX is illustrated in Figure 1.

4.1 Message Classifier

Commit Message Augmentation To address the issue of incomplete or empty commit messages, we design a multi-source contextual augmentation mechanism (Li et al. 2024). For each commit, we extract its associated issue and pull request (PR) identifiers and retrieve their corresponding descriptions and comments via the GitHub REST API. These additional texts serve as supplementary information to enrich the original commit message. The augmented commit message M_{ext} is constructed as:

$$M_{\text{ext}} = \text{Concatenate}(M_{\text{base}}, D_{\text{issue}}, C_{\text{issue}}, D_{\text{pr}}, C_{\text{pr}}) \quad (1)$$

where M_{base} denotes the original commit message, D_{issue} and D_{pr} are the issue and PR descriptions, C_{issue} and C_{pr} are their corresponding comments, and Concatenate$(\cdot)$ represents the concatenation of all available textual content.

Supervised Fine-tuning We adopt Qwen2.5-1.5B as the backbone model and append a fully connected classification head that maps the final-layer hidden states to a probability distribution over classes. Given an augmented commit message M_{ext}, we tokenize it into a sequence $T = (t_1, t_2, \ldots, t_L)$, where L is the sequence length. The model processes T and produces hidden states $H = (h_1, h_2, \ldots, h_L)$, where $h_i \in \mathbb{R}^d$ and d is the hidden dimension. We apply max pooling over H to obtain the contextual representation V_c, which is passed through the classification head and a sigmoid activation function to generate the prediction:

$$\hat{y} = \text{Sigmoid}(W_c V_c + b_c) \quad (2)$$

where $W_c \in \mathbb{R}^{1\times d}$ and $b_c \in \mathbb{R}^1$ are the weights and bias of the classification head, and $\hat{y} \in [0, 1]$ denotes the predicted probability that the commit is vulnerability-fixing. We label vulnerability-fixing commits as positive (class 1) and non-vulnerability-fixing commits as negative (class 0). For a batch of B training samples, with true labels $y_i \in \{0, 1\}$ and predicted probabilities $\hat{y}_i \in [0, 1]$, we optimize the model using the binary cross-entropy loss:

$$\mathcal{L}_{\text{SFT}} = -\frac{1}{B}\sum_{i=1}^{B}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \quad (3)$$

Minimizing this loss enables the model to learn discriminative features from commit messages for accurate vulnerability-fix identification.

4.2 Patch Classifier

File Selection Mechanism Entangled commits are common in VFCI tasks, where a single commit may address multiple issues or include changes serving different purposes (Nguyen et al. 2023). Treating all modifications in such commits as vulnerability-related introduces noise, compromising the accuracy and reliability of VFCI methods. To mitigate this, we design a file selector to filter out irrelevant changes. The selector operates in two stages: heuristic-based preliminary filtering and LLM-based refined filtering.

(1) Heuristic-based Preliminary Filtering. This stage applies predefined rules based on file paths, file names, and function naming conventions to identify and eliminate changes that are likely unrelated to vulnerability fixes. These rules serve as an efficient first-layer filter to exclude common sources of noise. Specifically:

- File path and naming exclusion: Files located in directories such as `test/`, `tests/`, or `testing/`, or files with names containing keywords like `_test`, `Test`, or `mock` are excluded, as they are typically associated with test code.
- Function naming exclusion: For code files, functions with names starting with `test_` or containing keywords such as `assert`, `setUp`, or `tearDown` are excluded, as these are characteristic of testing frameworks.
- Documentation and resource file exclusion: Files with extensions such as `.md`, `.txt`, `.rst` (documentation), and `.json`, `.xml`, `.yaml`, `.ini`, `.config` (configuration and resource files) are filtered out, as they generally do not contain core logic fixes.

(2) LLM-based Refined Filtering. After the heuristic stage, some entangled changes may remain undetected. To address this, we introduce Qwen2.5-Coder-7B to perform semantic-level filtering. This stage leverages the LLM's capability in understanding both natural language and code, incorporating the augmented commit messages generated in the Message Classifier. For each file retained after heuristic filtering, we construct a tailored prompt that provides sufficient contextual information (Giray 2023; White et al. 2023) to help the model associate the semantic content of the code changes with the intent expressed in the commit message. The LLM determines whether a code change directly implements the commit's described purpose. Only files judged as relevant by Qwen2.5-Coder-7B are retained, effectively removing unrelated noise from entangled commits.

Feature Extractor Prior studies primarily focus on function-level or file-level code changes. However, our empirical analysis reveals that line-level modifications often contain critical vulnerability-fixing logic, which coarse-grained representations may fail to capture. To address this, we introduce a line-level code change feature extractor based on CodeBERT and CNN, designed to capture fine-grained semantics and local patterns in code changes.

After the file selector identifies files directly related to the commit message, we process their corresponding code changes. Specifically, we extract sequences of added and deleted lines. For any given line sequence $L = (l_1, l_2, \ldots, l_K)$, we first encode each line l_i using CodeBERT to obtain a fixed-dimensional vector $e_i \in \mathbb{R}^D$. The resulting sequence of embeddings is $E = (e_1, e_2, \ldots, e_K)$, where $E \in \mathbb{R}^{K \times D}$.

We then apply a CNN to extract local contextual features across code lines. To capture patterns of different granularities, we employ convolutional filters of three heights $h \in \{2, 3, 4\}$, with the same width D as the CodeBERT output. For each filter size h, we use M filters. Each filter $W_{h,j} \in \mathbb{R}^{h \times D}$ (where $j = 1, 2, \ldots, M$) performs a one-dimensional convolution over the line dimension of E, producing a feature map $c_{h,j} \in \mathbb{R}^{K+h-1}$. We apply global max pooling over each $c_{h,j}$ to extract the most salient feature:

$$p_{h,j} = \max(c_{h,j}) \quad \text{for } j = 1, 2, \ldots, M$$

For each filter size h, we concatenate the pooled values into a vector:

$$P_h = [p_{h,1}, p_{h,2}, \ldots, p_{h,M}] \tag{4}$$

Finally, we concatenate the vectors from all filter sizes to form the complete line-level feature representation:

$$F = [P_2; P_3; P_4] \tag{5}$$

Using this method, both the added and deleted line sequences are encoded into fixed-size feature vectors of dimension $3 \times M$, which capture the key semantic changes at the line level.

Classification Model Construction The feature vectors derived from the added and deleted line sequences are denoted as F_{add} and F_{rem}, respectively. To capture semantic differences and interactions between the two types of changes, we adopt a feature fusion strategy that combines concatenation and element-wise multiplication:

$$F_{\text{fused}} = [F_{\text{add}}; F_{\text{rem}}; F_{\text{add}} \odot F_{\text{rem}}] \tag{6}$$

This produces a fused feature vector $F_{\text{fused}} \in \mathbb{R}^{3 \times M \times 3}$. The fused vector is passed through a fully connected layer, followed by a sigmoid activation function to compute the prediction probability:

$$\hat{y} = \text{Sigmoid}(W_f F_{\text{fused}} + b_f) \tag{7}$$

where $W_f \in \mathbb{R}^{1 \times (3 \times M \times 3)}$ and $b_f \in \mathbb{R}^1$ represent the weights and bias of the classification layer. The output $\hat{y} \in [0, 1]$ indicates the probability that the commit is vulnerability-fixing. The model is trained using the binary cross-entropy loss defined in Equation (3), consistent with the Message Classifier.

4.3 Ensemble Classifier

Although the Message Classifier focuses on textual information and the Patch Classifier captures structural changes, each model has its own limitations. The Message Classifier may misclassify due to vague or incomplete messages, while the Patch Classifier may struggle with complex or noisy code changes. To overcome these limitations and leverage complementary strengths, we design an Ensemble Classifier based on the AdaBoost algorithm (Ying et al. 2013), which integrates both modalities to achieve more robust and accurate predictions.

Given a training dataset $D = \{(x_i, y_i)\}_{i=1}^N$, where x_i represents the i-th commit sample (including commit message and code changes), and $y_i \in \{0, 1\}$ is the ground truth label (1 for vulnerability-fixing, 0 otherwise), we perform T boosting rounds. At each iteration $t = 1, 2, \ldots, T$, the trained Message Classifier f_M and Patch Classifier f_P produce probability outputs:

$$p_{M,i} = f_M(x_i), \quad p_{P,i} = f_P(x_i) \tag{8}$$

where $p_{M,i}, p_{P,i} \in [0,1]$ denote the predicted probabilities. These are converted to binary predictions using a threshold $\theta = 0.5$:

$$\hat{y}_{M,i} = \begin{cases} 1, & \text{if } p_{M,i} \geq \theta \\ 0, & \text{otherwise} \end{cases}, \quad \hat{y}_{P,i} = \begin{cases} 1, & \text{if } p_{P,i} \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

The weighted classification error for each base classifier is calculated as:

$$\epsilon_M = \sum_{i=1}^{N} w_i \cdot \mathbb{I}[y_i \neq \hat{y}_{M,i}], \quad \epsilon_P = \sum_{i=1}^{N} w_i \cdot \mathbb{I}[y_i \neq \hat{y}_{P,i}] \tag{9}$$

where $w_i = \frac{1}{N}$ is the initial sample weight, and $\mathbb{I}[\cdot]$ is the indicator function. The weight coefficients of the classifiers are computed as:

$$\alpha_M = \frac{1}{2} \log \left(\frac{1 - \epsilon_M}{\epsilon_M} \right), \quad \alpha_P = \frac{1}{2} \log \left(\frac{1 - \epsilon_P}{\epsilon_P} \right) \tag{10}$$

These coefficients reflect the relative contribution of each classifier. Sample weights are updated to emphasize misclassified instances:

$$w_i \leftarrow w_i \cdot \exp\left(\alpha_M \cdot \mathbb{I}[y_i \neq \hat{y}_{M,i}] + \alpha_P \cdot \mathbb{I}[y_i \neq \hat{y}_{P,i}]\right) \tag{11}$$

Weights are then normalized to ensure $\sum_{i=1}^{N} w_i = 1$. After T iterations, we aggregate predictions using the accumulated weights $\alpha_M^{(t)}$ and $\alpha_P^{(t)}$. The final ensemble classifier is defined as:

$$f_{\text{Ensemble}}(x) = \frac{\sum_{t=1}^{T} \left(\alpha_M^{(t)} \cdot f_M(x) + \alpha_P^{(t)} \cdot f_P(x) \right)}{\sum_{t=1}^{T} \left(\alpha_M^{(t)} + \alpha_P^{(t)} \right)} \tag{12}$$

where $f_{\text{Ensemble}}(x) \in [0,1]$ represents the predicted probability that commit x is vulnerability-fixing. This ensemble approach effectively combines macro-level semantics from commit messages and micro-level structure from code patches, enabling more reliable predictions in complex scenarios.

5 Experimental Settings

5.1 Experimental Settings

Dataset Construction To evaluate the effectiveness of VFCionX, we construct a VFCI dataset containing both vulnerability-fixing commits (VFCs) and non-vulnerability-fixing commits (non-VFCs). Previous studies show that C/C++ are among the most vulnerable programming languages (Wang et al. 2019, 2021, 2020). Therefore, we focus on C/C++ projects and collect data from five popular open source repositories (Torvalds 2025; GAPC 2025; ImageMagick 2025; mruby 2025; openssl 2025). To obtain VFCs, we extract CVE records published since January 2022 from the NVD database. We then identify corresponding commits by matching commit hashes or reference links provided by NVD within the target repositories, resulting in 2,509 VFCs. Given that non-security-related commits vastly outnumber VFCs in open source repositories, we randomly sample 22,121 non-VFCs from the same five repositories, ensuring that none are associated with any CVE entry in NVD. We divide the dataset into training sets (70%), validation sets (10%), and test sets (20%), while maintaining the class distribution of VFC and non-VFC in each subset. The detailed statistics of each subset are summarized in Table 1.

Baselines To comprehensively evaluate the performance of VFCionX, we compare it against three representative categories of baseline methods: rule-based approaches, SLMs, and LLMs. First, the rule-based method identifies VFCs by checking whether the commit message contains vulnerability-related keywords. To support this strategy, we manually analyze commit messages and compile a keyword list consisting of 89 terms, including both general security-related expressions and Linux-specific terms such as "oops" and "KASAN". Second, SLM-based methods rely on small pre-trained language models (e.g., RoBERTa (Liu et al. 2019) and CodeBERT (Feng et al. 2020)) combined with fine-tuning to perform VFCI. Representative approaches include VulFixMiner (Zhou et al. 2021a), VulCurator (Nguyen et al. 2022), and MiDas (Nguyen et al. 2023), which encode features from commit messages and code changes to train supervised classifiers. Finally, LLM-based methods leverage the strong generalization capabilities of LLMs in zero-shot and few-shot learning settings to identify VFCs. We evaluate several representative models, including Llama-3.1-8B (Meta 2025b), Qwen2.5-7B (Qwen 2025), GPT-4.5 (OpenAI 2025), and DeepSeek-v3 (DeepSeek 2025).

Metrics Following prior studies (Nguyen et al. 2022; Li et al. 2024; Ding et al. 2024), we use precision, recall, and F1-score as evaluation metrics for the VFCI task. These metrics are defined as follows:

- Precision (P): $P = \frac{\text{TP}}{\text{TP+FP}}$
- Recall (R): $R = \frac{\text{TP}}{\text{TP+FN}}$
- F1-score (F1): $F1 = \frac{2 \times P \times R}{P+R}$

Here, TP denotes the number of correctly identified VFCs, FP refers to non-VFCs incorrectly predicted as VFCs, and FN represents the VFCs that are missed by the model.

Experiment Details VFCionX consists of three core modules: the Message Classifier, Patch Classifier, and Ensemble Classifier. For the Message Classifier, we perform SFT for 20 epochs using a learning rate of 5e-5 and a batch size of 32. For the Patch Classifier, we use convolutional layers with three kernel sizes: 2×768, 3×768, and 4×768, where 768 matches the hidden size of the CodeBERT outputs. The stride is set to 1, and each kernel size uses 128 filters. During training, we set the batch size to 1 and apply gradient accumulation with 32 steps. The model is trained for 20 epochs with a learning rate of 2e-5. For the Ensemble Classifier, we fix the predictions of the Message and Patch Classifiers and set the number of AdaBoost iterations to 50. All experiments are conducted on a server equipped with an Intel(R) Xeon(R) Platinum 8462Y+ CPU and a single NVIDIA H100 GPU with 80 GB memory.

Repository	Train		Valid		Test	
	#VFCs	#non-VFCs	#VFCs	#non-VFCs	#VFCs	#non-VFCs
Linux kernel	1,449	14,388	161	1,599	395	4,005
GPAC	52	282	7	40	15	81
ImageMagick	186	539	27	77	53	154
mruby	17	107	2	15	5	31
OpenSSL	98	562	14	80	28	161

Table 1: Statistics of the constructed VFCI dataset.

5.2 Main Results

To evaluate the effectiveness of VFCionX in VFCI, we conduct systematic comparative experiments against a diverse set of representative baseline approaches. Table 2 systematically compares the performance of VFCionX against three representative categories of baseline methods on the VFCI task. Rule-based methods rely on detecting vulnerability-related keywords in commit messages and achieve an F1-score of only 21.29%, reflecting the inherent limitations of keyword matching. As discussed in introduction, due to the constraints of CVD practices and low-quality commit messages, many VFCs do not explicitly mention vulnerabilities, resulting in substantial false negatives.

Among SLM-based methods, VulFixMiner and MiDas yield moderate performance, primarily because they underutilize commit messages and lack mechanisms to handle noisy or entangled commits. VulCurator achieves the best performance among SLMs (F1: 69.59%) by incorporating contextual signals such as commit messages and associated issue reports. However, its accuracy is still limited by its inability to distinguish non-vulnerability-related changes in entangled commits.

In the zero-shot setting, LLM-based methods demonstrate high recall (e.g., GPT-4.5 achieves 92.15%) but generally suffer from low precision. This suggests that without task-specific tuning or prompt guidance, LLMs tend to over-predict the positive class, leading to a high false positive rate. When provided with few-shot examples, the performance of LLMs improves significantly. For instance, GPT-4.5 achieves an F1-score of 72.05%, confirming the value of in-context examples in guiding model predictions. Nonetheless, LLMs remain susceptible to noise from entangled and redundant code changes.

In contrast, VFCionX integrates the contextual understanding capabilities of LLMs, the efficient feature extraction of SLMs, and the robustness of ensemble learning. It achieves the highest overall performance with an F1-score of 81.47%, outperforming all baselines. Compared to the best-performing LLM (GPT-4.5 in few-shot, F1: 72.05%), VFCionX improves the F1-score by 9.42%, and surpasses the best SLM-based method (VulCurator, F1: 69.59%) by 11.88%. These results demonstrate that VFCionX effectively addresses the key challenges of incomplete commit messages and entangled code changes, offering a more practical and generalizable solution for identifying vulnerability-fixing commits.

5.3 Ablation Study

We conduct an ablation study by comparing different variants of VFCionX to assess the contribution of each component to the overall performance. Specifically, VFCionX-MC and VFCionX-PC denote the variants that use only the Message Classifier and only the Patch Classifier, respectively. For VFCionX-MC, we further remove the message augmentation module, denoted as w/o Message Extension. For VFCionX-PC, we additionally remove the file selection module, denoted as w/o File Selector.

The experimental results are presented in Table 3. VFCionX-MC achieves an F1-score of 79.53%, indicating that the classifier based on augmented commit messages plays a critical role in VFCI. When the message extension component is removed, the F1-score drops to 75.30%, highlighting the effectiveness of enriching commit messages by linking related issues and pull requests. Enhanced messages provide valuable contextual information that improves the discriminative capacity of the classifier, addressing the inconsistency in the quality of the original commit messages. VFCionX-PC yields a relatively low F1-score of 62.09%, suggesting that relying solely on code patch information is insufficient. Upon removing the file selector, the F1-score further decreases to 55.26%, validating the presence of entangled commits and the effectiveness of our mitigation strategy. The file selector filters out irrelevant code changes unrelated to the commit message, allowing the Patch Classifier to better focus on the core vulnerability-fixing logic. However, even with the assistance of advanced LLMs in file selection, the impact of noisy changes cannot be completely eliminated. The complete VFCionX model, which integrates the predictions of the Message Classifier and the Patch Classifier through an Ensemble Classifier, achieves the highest F1-score of 81.47%, outperforming each individual classifier. This demonstrates that the Adaboost-based ensemble strategy effectively combines the strengths of both classifiers, improving the model's generalization and robustness.

5.4 Sensitivity Analysis

We conduct a sensitivity analysis on two key hyperparameters: the maximum commit message length (*Max Msg Length*) in the Message Classifier and the maximum number of changed lines of code (*Max Changed LOC*, including both added and deleted lines) in the Patch Classifier. For inputs exceeding the threshold, we apply truncation. The results are presented in Figure 2 and Figure 3.

Category	Method	Precision(%)	Recall(%)	F1-score(%)
Rule-based	Keyword Matching	31.51	16.08	21.29
SLMs	VulFixMiner	40.21	56.20	46.88
	MiDas	48.35	59.31	53.28
	VulCurator	<u>71.62</u>	67.67	69.59
LLMs (zero-shot)	Qwen2.5-7B	25.13	82.92	38.57
	Llama-3.1-8B	24.86	84.77	38.45
	GPT-4.5	43.09	92.15	58.72
	DeepSeek-v3	40.86	91.14	56.42
LLMs (few-shot)	Qwen2.5-7B	42.08	85.94	56.50
	Llama-3.1-8B	42.72	88.61	57.65
	GPT-4.5	58.10	**94.83**	<u>72.05</u>
	DeepSeek-v3	56.25	<u>92.91</u>	70.07
SLM+LLM	VFCionX	**76.54**	87.09	**81.47**

Table 2: Performance of baselines and VFCionX. Best results are in **bold** and second best results are <u>underlined</u>.

Model	Precision(%)	Recall(%)	F1(%)
VFCionX-MC	74.72	85.01	79.53
w/o Message Extension	69.88	81.63	75.30
VFCionX-PC	55.26	70.85	62.09
w/o File Selector	49.67	62.27	55.26
VFCionX	**76.54**	**87.09**	**81.47**

Table 3: Ablation results of VFCionX and its variants.

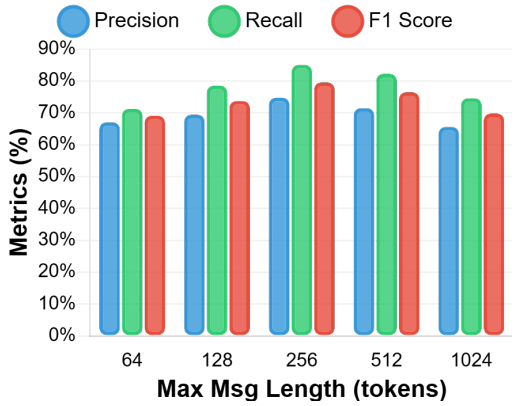

Figure 2: Impact of maximum message length on classification performance.

We observe that when the maximum commit message length is small, the model fails to capture sufficient contextual information, resulting in suboptimal performance. As the length increases, performance improves and peaks at 256 tokens. However, further increases lead to performance degradation, likely due to the inclusion of irrelevant content such as verbose debugging logs, which introduce noise into the learning process. For the maximum changed lines of code (LOC), we observe that the model performs better under lower thresholds (e.g., fewer than 10 lines). This aligns with prior findings that security patches are typically small, localized, and highly focused fixes (Zeng et al. 2021; Zhou et al. 2021b). As the LOC constraint increases, the input often includes a larger number of code modifications, which tend to involve unrelated changes such as test updates, formatting adjustments, or refactorings. These non-security-related edits introduce substantial noise, making it more difficult for the model to capture structural and semantic patterns indicative of vulnerability fixes, thereby degrading overall performance. This observation further confirms the adverse impact of commit entanglement on the VFCI task and underscores the importance of carefully constraining input boundaries in model design.

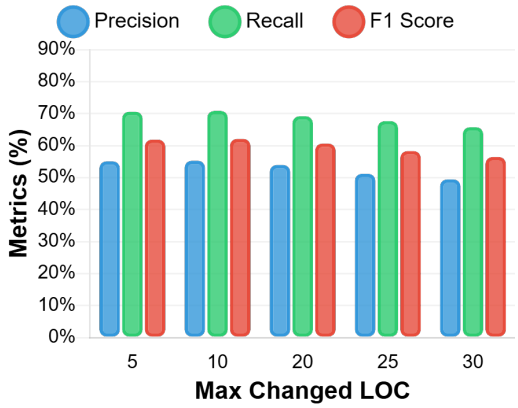

Figure 3: Impact of maximum changed lines of code (LOC) on classification performance.

6 Conclusion

This paper presents VFCionX, a novel approach to VFCI that combines the semantic understanding of LLMs with the efficient feature extraction of SLMs, enhanced by ensemble learning. To tackle noisy commit messages and entangled code changes, it introduces a context-aware Message Classifier and a noise-resistant Patch Classifier. Experiments on OSS projects show that VFCionX significantly outperforms existing methods in both accuracy and generalizability.

Acknowledgements

This paper is supported by the National Natural Science Foundation of China under No. 62202457, the Research Project of Quan Cheng Laboratory, China under Grant No. QCL20250107 and the YuanTu Large Research Infrastructure.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Boughton, L.; Miller, C.; Acar, Y.; Wermke, D.; and Kästner, C. 2024. Decomposing and measuring trust in open-source software supply chains. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 57–61.

DeepSeek. 2025. DeepSeek-V3. Https://huggingface.co/deepseek-ai/DeepSeek-V3.

Ding, Y.; Fu, Y.; Ibrahim, O.; Sitawarin, C.; Chen, X.; Alomair, B.; Wagner, D.; Ray, B.; and Chen, Y. 2024. Vulnerability detection with code language models: How far are we? *arXiv preprint arXiv:2403.18624*.

Dong, G.; Yuan, H.; Lu, K.; Li, C.; Xue, M.; Liu, D.; Wang, W.; Yuan, Z.; Zhou, C.; and Zhou, J. 2023. How abilities in large language models are affected by supervised fine-tuning data composition. *arXiv preprint arXiv:2310.05492*.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

GAPC. 2025. gpac source code. Https://github.com/gpac/gpac.

Giray, L. 2023. Prompt engineering with ChatGPT: a guide for academic writers. *Annals of biomedical engineering*, 51(12): 2629–2633.

Github. 2025. Commit 9805187. Https://huggingface.co/deepseek-ai/DeepSeek-V3.

Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

ImageMagick. 2025. ImageMagick source code. Https://github.com/imagemagick/imagemagick.

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. In Moschitti, A.; Pang, B.; and Daelemans, W., eds., *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1746–1751. Doha, Qatar: Association for Computational Linguistics.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213.

Ladisa, P.; Plate, H.; Martinez, M.; and Barais, O. 2023. Sok: Taxonomy of attacks on open-source software supply chains. In *2023 IEEE Symposium on Security and Privacy (SP)*, 1509–1526. IEEE.

Li, F.; and Paxson, V. 2017. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2201–2215.

Li, Y.; Zhang, T.; Widyasari, R.; Tun, Y. N.; Nguyen, H. H.; Bui, T.; Irsan, I. C.; Cheng, Y.; Lan, X.; Ang, H. W.; et al. 2024. CleanVul: Automatic Function-Level Vulnerability Detection in Code Commits Using LLM Heuristics. *arXiv preprint arXiv:2411.17274*.

Liu, C.; Chen, S.; Fan, L.; Chen, B.; Liu, Y.; and Peng, X. 2022. Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem. In *Proceedings of the 44th International Conference on Software Engineering*, 672–684.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Luo, G.; Zhou, Y.; Ren, T.; Chen, S.; Sun, X.; and Ji, R. 2023. Cheap and quick: Efficient vision-language instruction tuning for large language models. *Advances in Neural Information Processing Systems*, 36: 29615–29627.

Lyu, Q.; Havaldar, S.; Stein, A.; Zhang, L.; Rao, D.; Wong, E.; Apidianaki, M.; and Callison-Burch, C. 2023. Faithful chain-of-thought reasoning. In *The 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AACL 2023)*.

Meta. 2025a. CodeLlama-7B. Https://ollama.com/library/codellama:7b.

Meta. 2025b. Llama-3.1-8B. Https://huggingface.co/meta-llama/Llama-3.1-8B.

mruby. 2025. mruby source code. Https://github.com/mruby/mruby.

Nguyen, T. G.; Le-Cong, T.; Kang, H. J.; Le, X.-B. D.; and Lo, D. 2022. Vulcurator: a vulnerability-fixing commit detector. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1726–1730.

Nguyen, T. G.; Le-Cong, T.; Kang, H. J.; Widyasari, R.; Yang, C.; Zhao, Z.; Xu, B.; Zhou, J.; Xia, X.; Hassan, A. E.; et al. 2023. Multi-granularity detector for vulnerability fixes. *IEEE Transactions on Software Engineering*, 49(8): 4035–4057.

OpenAI. 2025. GPT-4.5. Https://openai.com/index/introducing-gpt-4-5/.

openssl. 2025. openssl source code. Https://github.com/openssl/openssl.

Perl, H.; Dechand, S.; Smith, M.; Arp, D.; Yamaguchi, F.; Rieck, K.; Fahl, S.; and Acar, Y. 2015. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 426–437.

Ponta, S. E.; Plate, H.; and Sabetta, A. 2020. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering*, 25(5): 3175–3215.

Qwen. 2025. Qwen2.5-Coder-7B. Https://huggingface.co/Qwen/Qwen2.5-Coder-7B.

Roziere, B.; Gehring, J.; Gloeckle, F.; Sootla, S.; Gat, I.; Tan, X. E.; Adi, Y.; Liu, J.; Sauvestre, R.; Remez, T.; et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Sabetta, A.; and Bezzi, M. 2018. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International conference on software maintenance and evolution (ICSME)*, 579–582. IEEE.

Torvalds. 2025. Linux kernel source code. Https://github.com/torvalds/linux.

Wang, X.; Sun, K.; Batcheller, A.; and Jajodia, S. 2019. Detecting" 0-day" vulnerability: An empirical study of secret security patch in OSS. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 485–492. IEEE.

Wang, X.; Wang, S.; Feng, P.; Sun, K.; Jajodia, S.; Benchaaboun, S.; and Geck, F. 2021. Patchrnn: A deep learning-based system for security patch identification. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, 595–600. IEEE.

Wang, X.; Wang, S.; Sun, K.; Batcheller, A.; and Jajodia, S. 2020. A machine learning approach to classify security patches into vulnerability types. In *2020 IEEE Conference on Communications and Network Security (CNS)*, 1–9. IEEE.

White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; and Schmidt, D. C. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*.

Wu, T.; He, S.; Liu, J.; Sun, S.; Liu, K.; Han, Q.-L.; and Tang, Y. 2023. A brief overview of ChatGPT: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5): 1122–1136.

Wu, Z.; Zhao, Y.; Wei, C.; Wan, Z.; Liu, Y.; and Wang, H. 2025. COmmitSHield: Tracking Vulnerability Introduction and Fix in Version Control Systems. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 279–290. IEEE.

Ying, C.; Qi-Guang, M.; Jia-Chen, L.; and Lin, G. 2013. Advance and prospects of AdaBoost algorithm. *Acta Automatica Sinica*, 39(6): 745–758.

Zeng, Z.; Zhang, Y.; Zhang, H.; and Zhang, L. 2021. Deep just-in-time defect prediction: how far are we? In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*, 427–438.

Zhou, J.; Pacheco, M.; Wan, Z.; Xia, X.; Lo, D.; Wang, Y.; and Hassan, A. E. 2021a. Finding a needle in a haystack: Automated mining of silent vulnerability fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 705–716. IEEE.

Zhou, Y.; and Sharma, A. 2017. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 914–919.

Zhou, Y.; Siow, J. K.; Wang, C.; Liu, S.; and Liu, Y. 2021b. Spi: Automated identification of security patches via commits. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1): 1–27.