EditFusion: Resolving Code Merge Conflicts via Edit Selection

Changxin Wang*, Lei Xu*†, Rundong Wang*, Yiming Ma*, Weifeng Zhang‡
*State Key Laboratory for Novel Software Technology, Nanjing University, China
‡School of Computer Science, Nanjing University of Posts and Telecommunications, China
*{502022330044, 522024330143, 502024330038}@smail.nju.edu.cn, †xlei@nju.edu.cn ‡zhangwf@njupt.edu.cn

***Abstract*—Merge conflicts in Distributed Version Control Systems (DVCS) like Git are a persistent challenge in the software development lifecycle. If not handled properly or overlooked, they can lead to issues like hindering collaboration and introducing errors. While automated resolution methods exist, prevailing approaches—such as multi-class classification and direct code generation—often suffer from limited interpretability, demanding substantial manual effort to refine predictions, and risk producing subtly flawed code. Critically, existing research often overlooks a prevalent conflict type: adjacent-line conflicts, where independent edits to contiguous lines are flagged by tools like Git. Our empirical analysis reveals that these make up a substantial portion of all conflicts. Moreover, they can often be resolved using simple patterns.**

Motivated by these limitations and empirical findings, we propose a novel approach: modeling merge conflict resolution as edit script selection. Instead of predicting abstract categories or generating code from scratch, our method makes a binary decision for each atomic line-level edit script contributing to the conflict: accept or reject. Our method inherently makes the reasoning behind proposed solutions transparent, as decisions directly correspond to individual, developer-authored code modifications. It also aligns closely with how developers naturally approach conflict analysis by considering each change in context. Our method applies to the vast majority (94.18%) of conflicts that can be correctly resolved without introducing any novel code lines—that is, lines not present in any of the parent versions being merged; this selection process directly yields the resolved code by applying the chosen subset of existing edits. We developed EDITFUSION, a deep learning model that performs edit script selection by leveraging semantic embeddings and edit metadata. Extensive evaluation on large-scale, real-world datasets demonstrates both the prevalence of adjacent-line conflicts and EDITFUSION's superior performance in accurately resolving conflicts compared to baselines. Our work represents an attempt towards more transparent, intuitive, and practical automated merge conflict resolution.

***Index Terms*—Code Merging, Automated Conflict Resolution, Edit Script, Adjacent-line Conflict**

I. INTRODUCTION

Distributed Version Control Systems (DVCS), such as Git, are foundational tools in modern software engineering, enabling efficient collaboration among developers, especially in large-scale, distributed, and open-source projects [1], [2]. The parallel nature of DVCS allows multiple developers to work concurrently on different features or fixes in separate branches. However, integrating these parallel changes frequently leads to merge conflicts, which occur when the DVCS cannot automatically reconcile concurrent modifications made to the same file regions [3], [4]. Manually resolving these conflicts is time-consuming and error-prone, with the potential to introduce new bugs [5]–[7] and delay project timelines.

Considerable research effort has been directed towards automating merge conflict resolution. Broadly, these efforts fall into two categories: improving traditional merge algorithms [8]–[14] and learning-based approaches predicting resolution strategies [15]–[18]. While advanced merge algorithms using structural (e.g., AST-based) or operational history information [19], [20] show promise, standard text-based three-way merging remains dominant in practice due to its generality and efficiency [21]. Consequently, predicting resolution strategies for conflicts generated by standard tools is a highly relevant research direction.

However, prevailing resolution prediction methods, often framed as multi-class classification tasks [15], [16], [22], suffer from several limitations. First, they often lack interpretability; developers are given a predicted category without a clear rationale based on the specific code changes involved. Second, for complex resolution categories, these models typically only provide the category label, still requiring significant manual effort from the developer to craft the actual code solution. Third, they may rely heavily on hand-crafted features, limiting adaptability across different languages or project contexts. Generative models [17], [18], [23], [24] offer an alternative by generating solution code directly, but can suffer from hallucination issues, potentially generating subtle, hard-to-detect syntactically correct but semantically flawed code, and often incur significant computational costs.

Furthermore, existing research often overlooks a prevalent type of conflict: *adjacent-line conflicts*. These arise when independent edits are made to contiguous, but not overlapping, code lines in the base version. Standard tools like Git often group these adjacent line edits into a single conflict hunk to avoid potentially breaking semantic dependencies [14], [25]. Our empirical analysis on 1,070,489 real-world conflicts reveals that adjacent-line conflicts constitute a substantial portion of all conflicts and are particularly dominant in certain resolution categories.

Crucially, a significant fraction of these adjacent-line conflicts can be correctly resolved simply by accepting *all* involved edits. This observation suggests that many conflicts

†Corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00055

have simpler resolution patterns tied to the specific edits involved, patterns which current classification models may not effectively capture or leverage. And this observation may inspire a new approach to conflict resolution that directly leverages the specific edits.

We propose a novel approach to automated merge conflict resolution based on *edit script selection*. Instead of predicting a high-level resolution category, we model the resolution process as a sequence of binary decisions: for each atomic line-level edit script contributing to the conflict, should it be accepted or rejected in the final merged code? This approach offers several advantages:

- **Interpretability:** The decision process directly relates to individual code changes, making the rationale behind a proposed solution transparent.
- **Developer Intuition Alignment:** It mirrors how developers often reason about conflicts–by considering each change in relation to others.
- **Direct Code Generation:** For conflicts with resolutions using only code from the parent versions (which our empirical study showed is the vast majority), 94.18% of them can be directly resolved by applying a subset of existing edit scripts.

Based on this modeling method, we designed and implemented EDITFUSION, a deep learning model specifically tailored for the edit script selection task. We subsequently evaluated its performance on a large-scale dataset we curated, alongside two datasets from existing studies. The complete implementation and replication package are publicly available online at https://github.com/EditFusion/EditFusion.

The main contributions of this paper are:

1) A novel approach for modeling merge conflict resolution based on binary selection of edit scripts, enhancing interpretability and enabling direct code generation for most conflict types.
2) The design and implementation of EDITFUSION, a deep learning model incorporating semantic embeddings and edit metadata to accurately predict edit script acceptance.
3) An extensive empirical evaluation on large-scale, real-world conflict datasets demonstrating the prevalence of adjacent-line conflicts and the performance of EDITFUSION on practical conflicts.

II. MOTIVATION

Fig. 1 illustrates a common merge conflict in collaborative software development. In this example, the TARGET branch modified a field and added an insertion, while the SOURCE branch modified an adjacent field and added another insertion. These edits—modifications on adjacent lines and insertions at the same location—are flagged as a conflict by *git-merge*.

To comprehensively investigate such adjacent-line conflicts [25] phenomenon, we first conducted a large-scale empirical study. We meticulously curated a dataset of more than 107,0489 merge conflicts using stringent filtering criteria (further detailed in Section V-A). Adopting a conflict classification

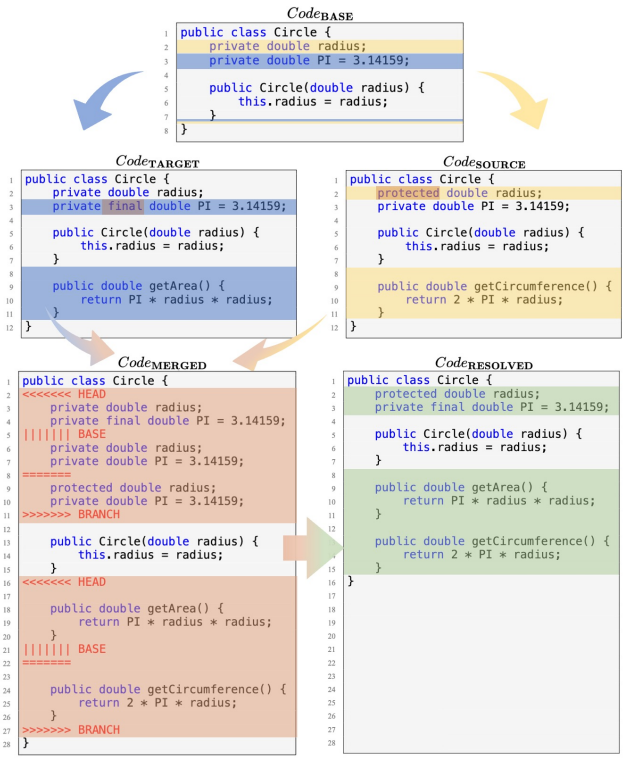

Fig. 1: A motivating example of a merge conflict.

scheme based on resolution patterns similar to previous studies [26], we classify merge conflicts into several categories:

- **V1 (*Version 1*)**. The resolution exclusively adopts all changes from the target branch;
- **V2 (*Version 2*)**. The resolution exclusively adopts all changes from the source branch;
- **CC (*ConCatenation*)**. The resolution is a concatenation of **V1V2** or **V2V1**;
- **CB (*ComBination*)**. The resolution is a cherry-picking of lines of three versions of code;
- **NC (*New Code*)**. The resolution primarily consists of newly written code lines that were not present in the conflicting sections of the parent branches or their common ancestor.

Through the empirical study (detailed in VI-A), we found that such adjacent edits are flagged as merge conflicts by Git and are widespread in practice. The analysis of our large-scale dataset revealed that **44.45%** of all conflicts fall into the category of adjacent-line conflicts. Notably, **97.01%** of CC conflicts and **82.73%** of CB conflicts are adjacent-line conflicts.

We further explored how many conflicts could be resolved by accepting a subset of all available edits to produce the correct solution code. By definition, all V1 and V2 conflicts can be resolved by accepting all edits from the respective branch and rejecting those from the other. Conversely, NC conflicts, which introduce new code, cannot be resolved solely

through existing edits. For other types (CC and CB), we employed a backtracking algorithm to enumerate and analyze all combinations of accepting or rejecting their constituent edits. Our results demonstrated that 74.99% of all conflicts can be correctly resolved by accepting a specific subset of the available edits. Notably, among all non-NC merge conflicts, this figure rises to 94.18%, indicating that the vast majority of conflicts not requiring new code can be resolved by selecting from existing edits. This led us to the realization that most conflicts can indeed be resolved by strategically choosing among the existing edits provided by developers.

However, existing automated resolution approaches, particularly those based on multi-class classification [15], [27] might predict a resolution strategy like *CB* or needing *Manual Edit* for the example in Fig. 1.

The developer is still required to:

1) Analyze the conflict hunks to understand the individual changes.
2) Manually write the code to resolve the conflict predicted as *Combine* or *Manual Edit*.

This gap—representing the persistent manual effort and cognitive burden still imposed on developers even with existing automated strategies—motivates our work. We propose a practical approach designed for production use with standard text-based merge tools, focusing directly on the **individual edits** that constitute a conflict, rather than predicting an abstract resolution category.

We term this the *edit selection* method. This approach enables the direct construction of resolutions for a large class of conflicts, including most adjacent-line scenarios like the one illustrated. Furthermore, by grounding the resolution process in decisions about specific, identifiable edits presented by standard tools, it inherently offers greater **interpretability** compared to abstract classifications—the reasoning behind a generated solution can be directly traced back to the decisions made about each conflicting change. This method is designed to be **minimally intrusive**, leveraging the output of existing diff tools, and ultimately aims to reduce manual effort.

III. Concepts

Definition 1 (Merging Commits)**.** A typical merge or rebase operation in Git involves three specific commits (versions) of the project's history:

- $Commit_{\text{TARGET}}$: The latest commit on the current branch (the target branch where changes are being merged into).
- $Commit_{\text{SOURCE}}$: The latest commit on the branch being merged.
- $Commit_{\text{BASE}}$: The most recent common ancestor commit of $Commit_{\text{TARGET}}$ and $Commit_{\text{SOURCE}}$. It represents the state from which both branches diverged.

Definition 2 (Edit Script)**.** An individual **edit script**, denoted es, represents an atomic, line-level transformation required to change one code version into another. It typically corresponds to the insertion, deletion, or modification of a contiguous block of lines.

Formally, we represent an edit script es with a tuple (os, oe, ms, me), where:

- $[os, oe)$ specifies the line range (start inclusive, end exclusive) in the *original* code version ($Code_1$).
- $[ms, me)$ specifies the corresponding line range in the *modified* code version ($Code_2$).

A diff algorithm, denoted Δ, is used to compute the differences between two code versions. The operation $\Delta(Code_1, Code_2)$ yields a set of such edit scripts, $\{es_1, es_2, \ldots, es_n\}$, representing the complete transformation. Specifically, in the context of a three-way merge (Definition 1), we consider two sets of edit scripts derived from the common ancestor $Code_{\text{BASE}}$: $\Delta_{\text{TARGET}} = \Delta(Code_{\text{BASE}}, Code_{\text{TARGET}})$ and $\Delta_{\text{SOURCE}} = \Delta(Code_{\text{BASE}}, Code_{\text{SOURCE}})$, representing the edits made on the target and source branches, respectively.

Definition 3 (Edit Script Intersection)**.** Two edit scripts, es and es', are considered to intersect in the original code space ($Code_{\text{BASE}}$) if their line ranges overlap or touch. Using the half-open interval $[start, end)$, intersection occurs if:

$$\min(oe, oe') \geq \max(os, os') \tag{1}$$

This definition identifies edits on adjacent lines as intersecting. For example, edits affecting base code ranges $[5, 6)$ and $[6, 7)$ respectively satisfy the intersection condition. This is crucial as it mirrors the behavior of *git-merge*.

Definition 4 (Adjacent-Line Conflict)**.** Merge conflict hunks are generated by *git-merge* when edit scripts from different branches intersect in $Code_{\text{BASE}}$. As established in the Definition of Edit Script Intersection, this condition includes edits that are merely on adjacent lines (i.e., they "touch"). An adjacent-line conflict is then a specific type of merge conflict where the involved edit scripts from Δ_{TARGET} and Δ_{SOURCE}, while fundamentally affecting contiguous but non-overlapping line ranges in the original $Code_{\text{BASE}}$, are nevertheless grouped into a single conflict hunk by Git, since their adjacency is treated as an intersection. Formally, a conflict generated from edits $es_1, \ldots, es_n$ constitutes an adjacent-line conflict if there exists a permutation $S = (es'_1, \ldots, es'_n)$ of these edits such that:

$$\bigwedge_{i=1}^{n-1} \Big((es'_i.oe = es'_{i+1}.os) \wedge (P(es'_i) \neq P(es'_{i+1})) \Big) \tag{2}$$

$P(es)$ denotes the branch origin of edit script es (either Δ_{TARGET} or Δ_{SOURCE}). The conjunction ($\wedge$) within the $\bigwedge_{i=1}^{n-1}$ quantifier ensures that for every consecutive pair of edits (es'_i, es'_{i+1}) in the sequence S, two properties must hold simultaneously:

- **Spatial Adjacency:** The first condition, $es_i.oe = es_{i+1}.os$, guarantees that the line range affected by es_i in the original code ($Code_{\text{BASE}}$) ends precisely where the range affected by es_{i+1} begins.
- **Alternating Origin:** The second condition, $P(es_i) \neq P(es_{i+1})$, guarantees that the two spatially adjacent edits originate from different branches.

Therefore, the entire formula requires the existence of a sequence S composed of spatially adjacent edits whose branch origins strictly alternate.

IV. Approach

A. Overall Workflow: Edit Script Selection

Our approach fundamentally shifts the focus from predicting high-level resolution categories to making decisions directly about the constituent **individual edits**. Fig. 2 outlines the overall workflow. This process is articulated through the following key stages:

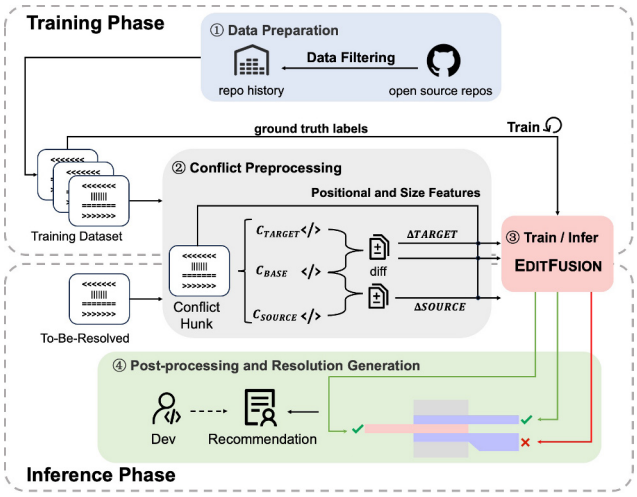

Fig. 2: Overall Workflow

Data Preparation. This process involves repository filtering, replaying merge scenarios to accurately identify conflicts, and extracting developers' ground truth resolutions (C_{RESOLVED}). Each conflict is then represented as a tuple: $(C_{\text{BASE}}, C_{\text{TARGET}}, C_{\text{SOURCE}}, C_{\text{RESOLVED}})$.

Conflict Preprocessing. A standardized pipeline first extracts the relevant edit scripts. For training data preparation, an additional filtering step uses a resolvability check algorithm to retain only conflicts where C_{RESOLVED} can be perfectly reconstructed from a subset of all edit scripts.

Model Training and Inference. Preprocessed sequences serve as input to the EditFusion model (architecture in Section IV-B). During the *training phase*, the model undergoes supervised learning to map an edit script's features and sequential context to its ground truth acceptance label. In the *inference phase*, the trained model predicts the acceptance probability for each edit script in a unseen conflict sequence.

Post-processing and Resolution Generation. During inference, this final stage constructs the recommended code resolution C'_{RESOLVED} by applying the accepted edit scripts to C_{BASE}. This resolution, accompanied by the explicit accept/reject decision for each contributing script, is then presented to developers to enhance their understanding and trust compared to black-box approaches.

B. Model Architecture

EditFusion, a deep learning model specifically for the task of predicting the acceptance or rejection of each edit script within a conflict sequence. The architecture, illustrated conceptually in Fig. 3, leverages Recurrent Neural Networks (RNNs) to effectively capture dependencies and contextual information within the ordered sequence of edits involved in a conflict.

The model processes the conflict through the following key stages:

1) **Edit Script Feature Embedding:** For each individual edit script es_i in this set, we compute a comprehensive feature embedding. This involves extracting both positional/size features and generating a semantic representation.
2) **Sequential Context Encoding (RNN):** The sequence of feature embeddings corresponding to the edit scripts $S = (es_1, \ldots, es_k)$ (ordered by their start position in $Code_{\text{BASE}}$) is then fed step-by-step into the core RNN layers. The RNN processes this sequence, leveraging its recurrent nature to capture dependencies and contextual information across the edits. For each input timestep i (representing es_i), the RNN outputs a context-aware hidden state that encodes information about es_i in relation to other scripts in the sequence.
3) **Classification:** Finally, the context-aware hidden state generated by RNN for each edit script es_i is passed through a classifier. This classifier is implemented as a linear layer followed by a *Sigmoid* activation function, which outputs a probability score between 0 and 1 for each es_i, representing the model's prediction of whether that specific edit script should be accepted into the final resolved code.

C. Features

To accurately predict the acceptance or rejection of an edit script within its conflict context, EditFusion relies on features that capture both the semantic meaning of the code change and its positional and size characteristics.

1) Code Change Semantic Embeddings: Effectively capturing the semantic intent and impact of each edit script is paramount for understanding its role in a merge conflict and deciding whether it should be included in the resolution. Simple lexical features often fail to grasp the nuances of code changes. Therefore, we employ a data-driven approach, leveraging large pre-trained language models specialized for code and fine-tuned for our specific task to represent code changes.

Our method draws inspiration from prior work LTRE [28], which demonstrated the value of using explicit, discrete sequence representations of code edits. LTRE represented a change using three sequences: the code before (x^-), the code after (x^+), and an token alignment sequence ($align(x^-, x^+)$) detailing the edit operations, feeding these into an LSTM encoder. However, this approach predated the widespread success of fine-tuning large transformer-based models.

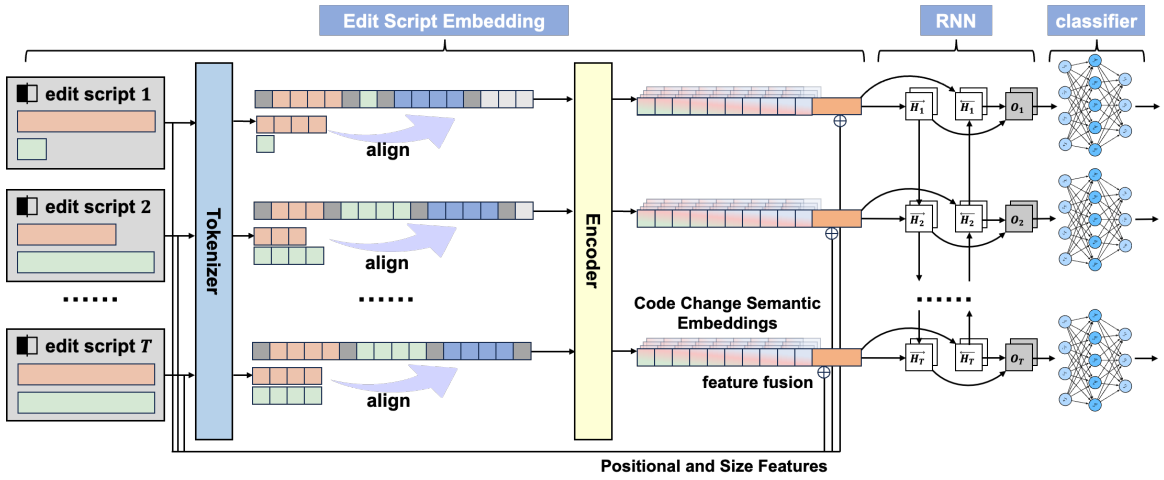

Fig. 3: Model Architecture of EDITFUSION.

We adapt this explicit representation strategy but integrate it with a more powerful, fine-tuned pre-trained code language model (details of the specific model are in Section V-D).

For the example edit script in Fig. 4 , we construct its semantic embedding as follows:

Tokenization: Code before and after modications are first tokenized using the tokenizer corresponding to the pre-trained language model. Let the token sequences be x^- and x^+, respectively.

```
1 - print("Hello, " + name) # comment
2 + print(f"Hello, {name} Welcome!")
```

Fig. 4: An example edit script.

Alignment Sequence Generation: A token-level diff algorithm is then used to compare x^- and x^+ and generate an alignment sequence, $align(x^-, x^+)$.

x^-	print	("	○	○	Hello	,	Ġ"	Ġ+	Ġname	○	○	)	Ġ#	Ġcomment
x^+	print	(	f	"	Hello	,	Ġ{	name	}	ĠWelcome	!	)	○	○
$align(x^-, x^+)$	=	↔	+	+	=	=	↔	↔	↔	+	+	=	−	−

Fig. 5: The code change alignment generated for the example edit script. Dark gray represents [CLS] or [SEP]. Gray represents the $\langle cc\_pad\_token \rangle$. Assuming a maximum length of 15, each sequence is padded to 15, displayed as white grids.

As shown in Fig. 5, this sequence explicitly captures the transformation at the token level. To encode the edit alignment directly within the input sequence for the language model, we introduce several special tokens into the model's vocabulary during the fine-tuning process.

- $\langle cc\_ins\_token \rangle$: Represents an inserted token (+).
- $\langle cc\_del\_token \rangle$: Represents a deleted token (−).
- $\langle cc\_rep\_token \rangle$: Represents a replaced token (↔).
- $\langle cc\_pad\_token \rangle$: A special padding token, distinct from the model's standard padding token, used during alignment (○).

The $align(x^-, x^+)$ sequence is constructed using these special tokens, providing a detailed edit trace.

Input Sequence Construction: Depending on the input format requirements of the pretrained models, we concatenate three sequences using the model's special tokens like [CLS] and [SEP](or <s> and </s>) to form a single input sequence. The standard structure is: [CLS] x^- [SEP] x^+ [SEP] $align(x^-, x^+)$ [EOS]. In this format, [CLS] acts as the beginning-of-sequence token whose final hidden state is used as an aggregate representation.

Embedding Extraction: This constructed input sequence is fed into the fine-tuned pre-trained code language model. We then extract the final hidden state vector corresponding to the initial [CLS] token directly from the output of the model's encoder layers.

This process produces a dense vector for each edit script, encapsulating its semantic meaning and ready for feature fusion.

2) Positional and Size Features: To complement the semantic information, we include meta features describing the positional and size properties of each edit script. With the prevalence of edit scripts that are adjacent to each other in the code, we believe that the position and size information can provide valuable context.

- Original start line: os
- Original end line: oe
- Modified start line: ms
- Modified end line: me
- Original length: $\Delta_{\text{pre}} = oe - os$
- Modified length: $\Delta_{\text{post}} = me - ms$
- Length difference: $\Delta_{\text{diff}} = \Delta_{\text{post}} - \Delta_{\text{pre}}$

These 7 numerical features capture the location and size impact of the edit in both original and modified versions. They

are normalized with min-max scaling before use.

3) Feature Fusion: The semantic embedding and the normalized positional and size feature vector are concatenated to form a single feature vector for each edit script.

V. Evaluation Setup

- **RQ1(Conflicts Characteristics): From the perspective of edit script selection, what relevant characteristics do real-world merge conflicts exhibit?** This RQ investigates characteristics that directly inform the feasibility and potential of our approach:
  (a) What percentage of conflicts can be resolved by selecting a subset of their original edit scripts?
  (b) What percentage of conflicts are classified as adjacent-line conflicts?
  (c) Among these adjacent-line conflicts, what proportion exhibits the simplest resolution pattern: being correctly resolved by accepting all involved edits?
- **RQ2(Comparative Effectiveness):** How effective is EditFusion in resolving entire merge conflicts compared to baseline methods? This RQ compares EditFusion against representative baseline models that implement alternative conflict resolution methodologies.
- **RQ3(Discrimination Performance):** How effectively does EditFusion discriminate between acceptable and unacceptable edit scripts? This RQ evaluates the model's core discriminative power at the edit script level.
- **RQ4(Ablation Study):** What is the contribution of the key components within the EditFusion model to its overall performance? This RQ investigates the impact of our design choices for EditFusion through ablation studies.
- **RQ5(Comparison with LLMs):** How does EditFusion compare against state-of-the-art Large Language Models? This RQ evaluates the effectiveness of EditFusion against prominent general-purpose LLMs.

A. Datasets

Investigating conflict characteristics (**RQ1**) required a large, representative dataset. We constructed one by systematically collecting and rigorously filtering merge conflict data from GitHub. The filtering criteria targeted repository activity (e.g., stars, recent updates), primary language diversity, and collaborative health (e.g., non-fork, organization-maintained, contribution patterns) to ensure representativeness across over 23,000 repositories. This process yielded over one million conflict instances for empirical analysis.

For a fair comparison against prior work (**RQ2**), when comparing EditFusion with baseline models, we conducted experiments on the datasets adopted in the original papers of those respective baselines.

Limited by cost and time, when comparing with SOTA LLMs (**RQ5**), we selected a dataset of 1,400 conflict code files, containing a total of 6,110 conflict code hunks including V1 (2,055), V2 (2,007), CB (1,121), CC (285), and NC (642).

B. Baselines

To evaluate the effectiveness of EditFusion relative to existing techniques (**RQ2**), we selected the following representative baseline methods for comparison:

- **MergeBERT** [16]: As a state-of-the-art classification-based method, MergeBERT addresses line-level conflicts from Git by first attempting a token-level merge. It employs a nine-class classification model to predict the resolution. A key innovation is its "merge tuple aggregation" method, which unifies all features related to a conflict hunk into a single representation. This contrasts with our edit-script-centric approach, where the semantic embedding is generated for each individual change (concerning only two parts, text before/after editing), avoiding the need for complex fusion strategies across all three versions (C_{BASE}, C_{TARGET}, and C_{SOURCE}). We include it as a key baseline to compare the effectiveness of our line-level edit script selection approach against its token-level classification paradigm.
- **DeepMerge** [29]: This approach represents sequence-to-sequence models for conflict resolution operating at the line level. It employs a Pointer Network [30] trained to select and order lines from the base (C_{BASE}), target (C_{TARGET}), and source (C_{SOURCE}) versions to directly construct the resolved code (C_{RESOLVED}). We include DeepMerge as a baseline for comparing resolution construction capabilities, particularly on conflicts that involve combining existing lines rather than just selecting one side.
- **RPredictor** [15]: Representing traditional classification approaches, RPredictor utilizes various handcrafted features derived from the conflict context and project history. It employs a machine learning classifier to predict one of three high-level resolution strategies: accept the TARGET version, accept the SOURCE version, or requires Manual Edit.
- **Git_DIRTY**: This baseline originated from our empirical investigation process used to address RQ1. It represents a minimally modified version of the standard `git-merge` source code. The sole modification implemented was to alter the handling of adjacent-line edits specifically: instead of generating a conflict hunk as standard Git does, Git_DIRTY automatically accepts edits from both branches that are adjacent and concatenates them sequentially. This baseline benchmarks the simple *accept all adjacent edits* heuristic.

C. Evaluation Metrics

We assess the performance of EditFusion and baseline methods using metrics defined at two distinct levels of granularity, reflecting different aspects of the resolution process:

Conflict-Level Metrics. To evaluate the end-to-end effectiveness in resolving entire conflicts, our primary metric is **Exact Match Accuracy**. This measures the percentage of conflict instances for which the generated resolved code C'_{RESOLVED} is identical to the ground truth resolution C_{RESOLVED}.

Edit Script-Level Metrics. To evaluate the core predictive capability of EDITFUSION in making its fundamental accept/reject decisions (relevant to RQ3), we analyze its performance at the individual edit script level. Treating this as a binary classification task for each script, we employ a suite of standard and robust metrics: **Precision**, **Recall**, **F1-score**, **ROC-AUC** [31] (Area Under the Receiver Operating Characteristic Curve), **Matthews Correlation Coefficient (MCC)** [32], a robust metric that provides a balanced measure of classification quality even with imbalanced classes, and **Cohen's Kappa** [33], which assesses the agreement between predicted and actual classifications while accounting for the possibility of agreement occurring by chance.

D. Implementation

Based on the general architecture described in Section IV-B, we implemented EDITFUSION using specific components and configurations chosen through experimentation and informed by related work. For generating the semantic embeddings of code changes (Section IV-C1), we employed **CodeBERTa-small-v1** [34]. This 84M parameter RoBERTa-based model, pre-trained on a vast corpus of code, was fine-tuned using the explicit change representation sequence described earlier to effectively capture code edit semantics.

The backbone Recurrent Neural Network responsible for processing the sequence of fused feature vectors was implemented as a **2-layer Bidirectional Long Short-Term Memory** network. Each LSTM layer contained **256 hidden units** per direction. To mitigate overfitting during training, **Dropout** with a probability of 0.3 was applied between the Bi-LSTM layers. The final fully connected layer transforms the contextualized representation from the Bi-LSTM for each edit script into a single output value.

For training, we utilized the Adam optimizer and employed differential learning rates, applying a lower rate (e.g., 10^{-6}) to the pre-trained CodeBERTa parameters and a higher rate (e.g., 10^{-4}) to the randomly initialized Bi-LSTM and fully connected layers. A weighted Binary Cross-Entropy (BCE) loss function was used to counteract class imbalance between accepted and rejected scripts. This loss, $\mathcal{L}$, is defined as:

$$\mathcal{L} = -\frac{1}{M}\sum_{i=1}^{M}[w \cdot y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

where M is the batch size, $y_i \in \{0,1\}$ represents the true label for the i-th script, $\hat{y}_i$ is the model's predicted probability that the i-th script should be accepted, and w is the weight assigned to the accepted class (positive class) to address the imbalance. This loss function was coupled with a StepLR learning rate scheduler for stable convergence.

VI. RESULTS AND ANALYSIS

A. RQ1: Conflict Characteristics and Edit Selection Potential

This section investigates the properties of real-world conflicts through the lens of edit script selection.

First, regarding the **theoretical ceiling (a)**, we found that a vast majority of conflicts, specifically **94.18%** of non-NC conflicts, which adds up to **74.99%** of all conflicts could be perfectly resolved by selecting an appropriate subset of the conflicting edit scripts and applying them to the base version (C_{BASE}). This high percentage strongly suggests that most conflict resolutions fundamentally involve deciding which existing changes to keep or discard, rather than requiring complex code synthesis.

Second, turning to **adjacent-line conflicts (b)**, our analysis revealed their significant prevalence, which is particularly noteworthy given the common misconception that all conflicts are caused by overlapping edits. Across the dataset, **44.45%** of **all** encountered conflict instances fit the definition of an adjacent-line conflict (Definition 4). Especially, **97.01%** of CC-type conflicts and **82.73%** of CB-type conflicts were classified as adjacent-line conflicts.

Third, examining the **simplicity of adjacent-line conflicts (c)** identified in (b), we found **49.30%**, representing **21.91%** of **all** conflicts had a ground truth resolution identical to simply accepting and sequentially applying *all* involved edit scripts from both branches. This proportion was particularly high for specific categories, reaching **84.41%** within CB-type and **95%** within CC-type adjacent-line conflicts. This indicates that nearly half of adjacent-line conflicts follow a very simple *accept all* pattern.

Implication: These findings collectively underscore the relevance of the edit script selection approach. The high theoretical ceiling confirms its wide applicability, while the prevalence and frequent simplicity of adjacent-line conflicts highlight a promising opportunity for automatically resolving merge conflicts through edit selection.

B. RQ2: EDITFUSION Resolution Performance

This section assessed the overall effectiveness of EDITFUSION in resolving entire conflicts compared to representative baseline methods using Exact Match Accuracy.

Comparison vs. State-of-the-art (MERGEBERT). We evaluated EDITFUSION against MERGEBERT [16] on the dataset from the original MergeBERT paper but excluded conflicts where the ground-truth resolution involved new code (NC) for EDITFUSION's edit selection paradigm is not designed to generate novel code.

The detailed results of this head-to-head comparison are presented in Table I. Overall, EDITFUSION achieves an Exact Match Accuracy of **70.10%**, significantly outperforming MERGEBERT's accuracy of 58.15%. Notably, MERGEBERT's accuracy on CC-type conflicts is only **1.30%**, whereas EDITFUSION achieves **87.07%**. Upon analysis, we found that CC conflicts often involve insertions from both developers at the same location (e.g., adding two different functions). In these cases, MergeBERT's token-level merge attempts to align and combine matching tokens (such as function keywords) of two unrelated insertions while treating the non-matching parts as a large, fragmented conflict. This process fractures the holistic semantic intent of the original edits, leading to an incorrect resolution.

TABLE I: Comparison with MERGEBERT on its dataset (NC excluded)

Model	Overall Acc.	V1 Acc.	V2 Acc.	CB Acc.	CC Acc.
MERGEBERT	58.15%	67.02%	62.68%	47.95%	**1.30%**
EDITFUSION	**70.10%**	**70.96%**	**73.92%**	**61.13%**	**87.07%**

In contrast, by operating on the level of whole edit scripts, EDITFUSION preserves the semantic integrity of each change and excels on CC-type conflicts, which are predominantly composed of adjacent-line conflicts. It confirms that our script-selection offers a simpler, more straightforward decision process, which also leads to higher resolution accuracy.

Comparison vs. Sequence-to-Sequence (DEEPMERGE). As shown in Table II, when evaluated on the non-trivial conflict subset targeted by DEEPMERGE, EDITFUSION achieved a Top-1 Exact Match Accuracy of **73.08%**. This represents a substantial improvement of over 36 percentage points compared to the reported 36.50% Top-1 accuracy of DEEPMERGE [29]. Even considering DEEPMERGE's Top-3 accuracy (43.23%), EDITFUSION's Top-1 performance remains significantly higher, demonstrating a superior capability in constructing the precise correct resolution. Analyzing performance on specific resolution types within this dataset, EDITFUSION excels at CC conflicts with 87.07% accuracy, vastly outperforming both DEEPMERGE (44.40%) and the Git_DIRTY heuristic (56.72%). For CB conflicts, EDITFUSION (61.12%) also significantly improves upon DEEPMERGE (29.03%).

As shown in Table II, the simple Git_DIRTY heuristic achieves the highest accuracy specifically on Combination conflicts (63.16%). This finding aligns directly with our empirical results from **RQ1**, which revealed that CB conflicts have a particularly high proportion of adjacent-line edits, many of which follow the simple *accept all* resolution pattern. Since Git_DIRTY implements exactly this naive *accept all adjacent* strategy. Its strong performance on this subset is expected when the heuristic matches the ground truth. Conversely, Git_DIRTY's relatively lower accuracy on Concatenation conflicts (56.72%) occurs because its fixed sequential concatenation order (e.g., always V1V2) inherently fails when the ground truth requires the alternative V2V1 order. EDITFUSION (87.07%), however, demonstrates high accuracy on CC conflicts because it focuses on the individual acceptance of each conflicting edit script. The edit selection method considers a resolution successful if the correct set of constituent scripts is identified; this inherently allows for the construction of both V1V2 and V2V1 concatenation orders.

TABLE II: Comparison with DEEPMERGE

Metric	DEEPMERGE	Git_DIRTY	EDITFUSION
Acc Top-1	36.50%	63.02%	**73.08%**
Acc Top-3	43.23%	N/A	N/A
Acc (CC)	44.40%	56.72%	**87.07%**
Acc (CB)	29.03%	**63.16%**	61.12%

Comparison vs. Classification (RPREDICTOR). Table III presents the comparison result with RPREDICTOR on the Ghiotto dataset [26]. On this dataset, EDITFUSION achieved an overall Exact Match Accuracy of **53.74%**. For a fair comparison with RPREDICTOR [12], we use its reported *within-project* performance, as EDITFUSION's training and evaluation setup does not strictly enforce project disjointness between training, validation, and test sets, making the within-project setting more analogous. RPREDICTOR achieved 35.70% accuracy in this setting on its three-class prediction task (*V1*, *V2*, *Manual Edit*).

EDITFUSION achieves an Exact Match Accuracy of approximately 53.74%, in stark contrast to RPREDICTOR's 35.70%. This significant performance gap is particularly noteworthy because RPREDICTOR's accuracy includes correctly predicting the *Manual Edit* category. Furthermore, EDITFUSION demonstrates superior performance in resolving conflicts that simply require selecting one side, achieving a V1 Recall of **72.19%** (compared to RPREDICTOR's 62%) and a V2 Recall of **62.90%** (compared to RPREDICTOR's 47%).

TABLE III: Comparison with RPREDICTOR

Metric	Git_DIRTY	RPREDICTOR-*within*	EDITFUSION
Accuracy	13.22%	35.70%	**53.74%**
V1 Recall	5.34%	62%	**72.19%**
V2 Recall	4.56%	47%	**62.90%**

Implication: Across these comparisons on different benchmark datasets, EDITFUSION consistently outperforms representative baselines employing different methodologies. This validates the effectiveness of the edit script selection method for achieving high resolution accuracy.

C. RQ3: EDITFUSION Script-Level Prediction Accuracy

This section assesses EDITFUSION's core capability: accurately predicting the acceptance or rejection of individual edit scripts. As traditional conflict resolution baselines are not directly comparable at this granular script-selection level, we designed a heuristic baseline, **NAIVEA**. This baseline mimics a Zero-R strategy by unconditionally accepting all conflicting edit scripts. NAIVEA is expected to perform well on conflicts that are simply resolved by accepting all changes, a scenario favored by the characteristics of the Dinella dataset [29], which has a high proportion of adjacent conflicts and an approximate 5:2 ratio of accepted to rejected scripts in its resolutions. We choose this dataset because this inherent data imbalance also presents a challenge for EDITFUSION, which we addressed using a weighted loss function during training.

The comparative results are presented in Table IV. At the crucial edit script level, EDITFUSION demonstrates robust discriminative power, achieving an F1-score of 0.92. This strong F1-score is supported by a high recall of 0.91, similar to the NAIVEA baseline (1.00), but with a substantially improved precision of 0.92 (compared to NAIVEA's 0.71), indicating that EDITFUSION makes far fewer false positive acceptances. Furthermore, its excellent Matthews Correlation Coefficient

(MCC) and Cohen's Kappa (κ) scores, both achieving 0.71, underscore EDITFUSION's stable performance and strong classification capability, even on this imbalanced dataset. These values signify a high degree of reliability in its script classification and a strong agreement between the model's predictions and the actual ground truth labels.

Further analysis of extreme resolution patterns highlights EDITFUSION's strengths. The model maintains very high accuracy (95.57%) on *All True* conflicts where all scripts are accepted. Critically, EDITFUSION also achieves a notable **62.13%** accuracy on *All False* conflicts, where the correct resolution is to reject all contributing edit scripts (i.e., C_{RESOLVED} is identical to C_{BASE}). This performance on these cases is particularly significant given their rarity: they constitute only 3.45% of CB-type conflicts and a mere 0.18% of all conflicts. EDITFUSION's ability to correctly resolve these infrequent, minority-pattern conflicts underscores the effectiveness of its learned semantic understanding in robustly handling imbalanced scenarios and identifying when no proposed changes are appropriate.

Implication: EDITFUSION demonstrates strong and reliable performance in the core task of classifying individual edit scripts for acceptance or rejection. It shows the model's proficiency in handling imbalanced data and making nuanced decisions for individual scripts. This validates the effectiveness of its learned, context-aware approach at this crucial script level.

TABLE IV: Comparison on the Edit Script Level

Metric	NAIVEA	EDITFUSION
Edit Script-Level Classification		
Precision	0.71	**0.92**
Recall	**1.00**	0.91
F1-score	0.83	**0.92**
MCC	0.00	**0.71**
Kappa (κ)	0.00	**0.71**
Conflict-Level Resolution Precision		
All True	**100.00%**	95.57%
All False	0.00%	**62.13%**

Note: "All True" refers to conflicts where all edit scripts are accepted in the ground truth; Äll Falseërefers to conflicts where all are rejected.

D. RQ4: Contribution of EDITFUSION Components

We systematically altered or removed features and architectural choices, measuring the impact on both conflict-level resolution precision (of all resolvable conflicts) and script-level classification accuracy. Detailed results are presented in Table V and Fig. 6.

The key findings confirm the positive contribution of each tested element:

- **Positional and Size Features:** Removing positional and size features led to a substantial decrease in performance with conflict accuracy dropped from 81.14% to 72.95% and script accuracy from 88.03% to 83.78%. This highlights the critical importance of incorporating spatial context alongside semantic information for effective script selection.
- **Embedding Fusion Strategy:** To isolate the impact of embedding method, we replaced our concatenation-based approach with the strategy used in MergeBERT, where edit type embeddings are separated and linearly added to the code embeddings. This change resulted in a slightly lower ROC-AUC of 0.957, compared to 0.964 with our method. This suggests our simpler fusion strategy is marginally more effective and, more importantly, confirms that the embedding method is not the source of the large performance gap between EDITFUSION and MERGEBERT, reinforcing that our script-level framework is the key contributor.
- **Semantic Embedding Model:** Replacing the code-specialized **CodeBERTa** embeddings with those from the generic (but similarly sized) DistilRoBERTa [35] model resulted in lower accuracy. This validates the benefit of leveraging language models pre-trained specifically on source code for this task.
- **Architecture:** Using **GRU** units instead of the chosen Bi-LSTM architecture yielded slightly lower performance (e.g., conflict accuracy decreased to 78.36%), suggesting that the Bi-LSTM's capacity for capturing bidirectional context provides an advantage.
- **Weighted Loss Function:** Disabling the loss weighting, designed to handle the dataset's accept/reject imbalance, resulted in a small drop in conflict precision (to 80.70%) and a more noticeable decrease in metrics sensitive to imbalance like ROC-AUC (from 0.964 to 0.924), confirming its utility.

Overall, the full EDITFUSION configuration consistently outperformed all ablated variants across the evaluated metrics.

Implication: The ablation study validates our design choices for EDITFUSION. Its strong performance is not attributable to a single component but rather stems from the synergy between using code-specific semantic embeddings, crucial positional and size features, an effective recurrent architecture (Bi-LSTM), and appropriate handling of data imbalance through weighted loss.

TABLE V: Ablation Study Results

Variant	Script Acc.	Conf. Prec.	ROC-AUC
EF	**88.03%**	**81.14%**	**0.964**
EF − Positional/Size Feats	83.78%	72.95%	0.924
EF ↔ MergeBERT-style Emb	87.80%	80.80%	0.957
EF ↔ GRU	86.71%	78.36%	0.928
EF ↔ DistilRoBERTa	86.11%	77.31%	0.908
EF − Weighted Loss	87.90%	80.70%	0.902

E. RQ5: Comparison with Large Language Models

This section evaluated four representative LLMs: Gemini-1.5-pro, GPT-4o, DeepSeek-v3.1 and Qwen-plus-2025-01-25. We designed two distinct prompting strategies to assess their capabilities under different conditions:

- **Vanilla Prompting:** Directly input the generated conflict code files into the LLM, and require it to directly generate

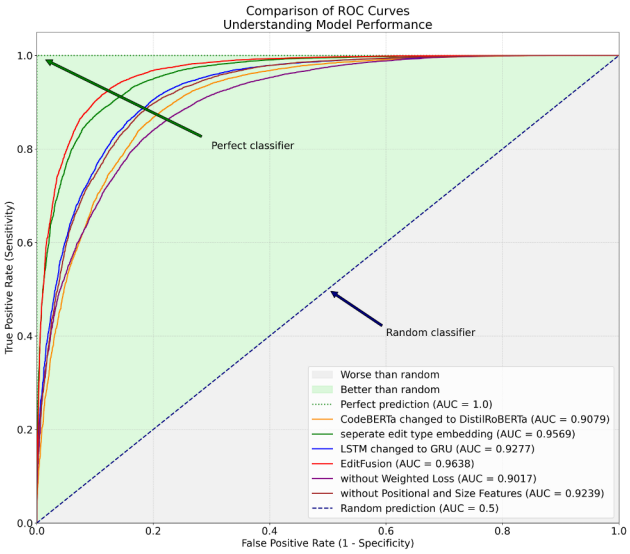

Fig. 6: Ablation Study Results

the merge results of the conflict parts, simulating the standard code merging prompting process.

- **Edit-Script-Aware Prompting:** Extra-structured edit scripts from ours and theirs versions to the base version are provided in the input, along with a brief explanation, to enhance the model's understanding of the semantics of the changes.

Table VI presents the performance of different models under both vanilla and edit-aware prompting strategies. We report results using three metrics—Exact Match Accuracy (Acc), BLEU, and normalized Edit Distance (Ed).

TABLE VI: Comparison with Large Language Models (LLMs)

Model	**Vanilla**			**Edit-Aware**		
	Acc	BLEU	Ed	Acc	BLEU	Ed
Gemini-1.5-pro	40.52%	0.3788	0.4715	**44.70%**	**0.3909**	**0.4576**
GPT-4o	24.98%	0.2633	0.6026	19.82%	0.2156	0.6720
DeepSeek-v3.1	27.61%	0.3330	0.5461	27.14%	0.3247	0.5678
Qwen-plus	19.72%	0.2411	0.6359	21.11%	0.2406	0.6505
EDITFUSION: **54.05%** (Match), **0.4823** (BLEU), **0.2043** (Ed)						

As shown in Table VI and Figure 7, LLMs handle simpler conflicts (V1 and V2) relatively well, but their accuracy drops sharply on complex cases (CC and CB). Among them, Gemini performs best, yet still falls short of EditFusion. BLEU and normalized edit distance metrics show the same pattern: LLMs may occasionally capture semantic similarity, but their outputs deviate more from developer resolutions, especially on harder conflicts. By contrast, EditFusion consistently achieves higher accuracy (54.05% vs. Gemini's 44.70%), better BLEU, and much smaller edit distances, reflecting closer alignment with ground truth.

Implication: The results further reveal that current LLMs cannot deliver stable accuracy, and EditScript-based prompting does not yield consistent improvements. The added input length and complexity likely destabilize most models, with Gemini showing only minor gains due to its stronger long-text modeling ability [36].

While LLMs are powerful in general, for automated merge conflict resolution where precision and reliability are critical, EDITFUSION's discriminative approach demonstrates greater robustness. This demonstrates that generative LLMs struggle to produce verbatim resolutions, whereas EditFusion's discriminative design ensures both precision and closer alignment to developer edits.

VII. DISCUSSION

A. Threats to Validity

The primary concern for **External Validity** is the objectivity and representativeness of the datasets used. The extraction of ground truth resolutions (C_{RESOLVED}) can, in some instances (e.g., edits outside conflict markers or fuzzy hunk boundaries), be challenging and potentially introduce minor inaccuracies. Furthermore, any single dataset might possess inherent biases. To mitigate these threats, we (1) constructed a new, large-scale dataset comprising over one million conflicts from more than 23,000 repositories, employing rigorous filtering to enhance its representativeness, and (2) evaluated EDITFUSION across this new dataset alongside two existing public benchmark datasets, where we observed consistent core performance trends, bolstering the generalizability of our findings.

Regarding **Internal Validity**, given that our edit script selection method is particularly pertinent to adjacent-line conflicts, its assessment relies on the consistent behavior of Git in forming such conflicts. Different Git versions, underlying diff algorithms (e.g., Myers, histogram), or merge strategies (e.g., recursive, ort) could potentially alter how adjacent edits are presented or grouped by the version control system. To address this, we conducted experiments across multiple Git implementations (C Git, JGit), various major Git versions (spanning two decades), and different diff/merge strategies. Our findings suggest that our conclusions are robust and not artifacts of a specific Git configuration.

B. Limitations

Despite the promising performance of EDITFUSION demonstrated in practice, our study acknowledges certain limitations. Primarily, EDITFUSION's effectiveness may be constrained when applied to programming languages, Domain-Specific Languages (DSLs), or file types not well-represented in the pre-training data of its underlying model, as their unique syntactic or semantic features might not be fully captured. Furthermore, our current approach resolves individual conflict hunks in isolation. It does not explicitly model or address more complex scenarios involving semantic dependencies that may span multiple conflict hunks within the same file or across different files involved in a single merge, potentially challenging its performance in such inter-connected conflict situations.

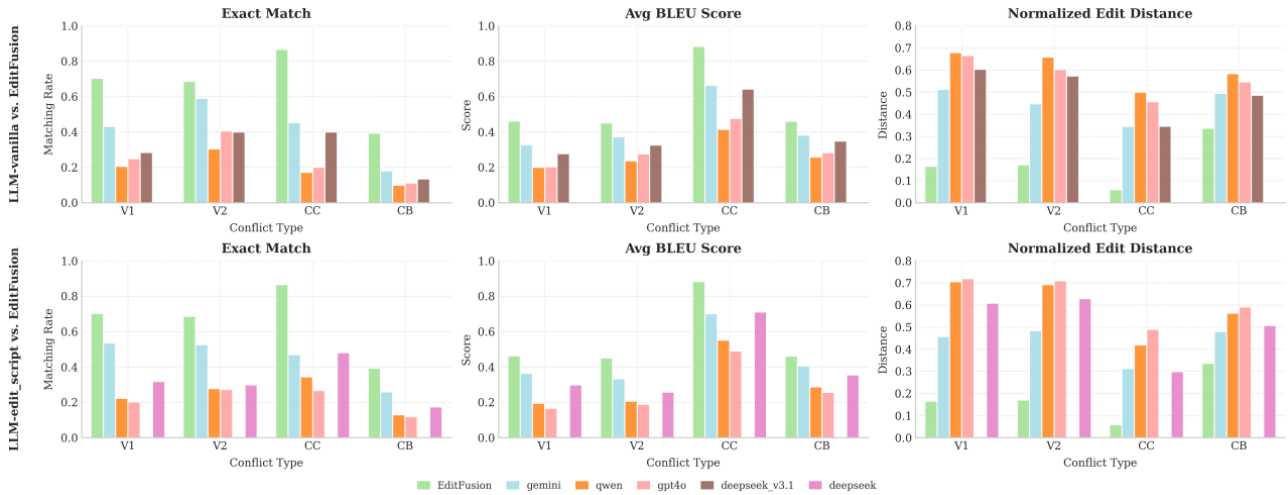

Fig. 7: Performance Across Conflict Types by Prompt Strategy

VIII. RELATED WORK

A. Traditional Merge Algorithms

While standard text-based three-way merge is widely used due to its generality, its lack of code structure understanding often leads to unnecessary conflicts or incorrect resolutions for semantically simple changes.

Structured and Semi-Structured Merging techniques attempt to address this by incorporating syntactic or semantic information. Structured approaches like JDime [8] operate on Abstract Syntax Trees (ASTs), comparing and merging code at the level of program elements. This allows for more precise conflict detection and can handle order-insensitive changes or refactorings. However, their adoption in practice has been hindered by several factors. These include high computational complexity [37], as operations on ASTs may involve solving NP-hard problems like the Tree Amalgamation Problem and the Maximum Common Embedded Subtree Problem [38], [39]. Further challenges are language-dependency (requiring specific parsers for each language) and difficulties in preserving original formatting or comments [21]. Semi-structured merging, exemplified by FSTMerge [10], jsFSTMerge [40] and IntelliMerge [9], offers a hybrid solution [41]. These methods typically parse high-level code structures like classes or functions to guide the merge process, while potentially reverting to text-based merging for the content within these structures. This aims to balance the precision of structured merging with the efficiency and generality of text-based approaches.

B. Learning-Based Approaches

Recent research increasingly leverages machine learning to address code conflicts, focusing on predicting or generating resolutions. These methods leverage large datasets to understand the underlying semantics of conflicts, thereby informing the proposal of solutions. The resulting resolution techniques can be broadly categorized:

Classification-Based Solution Prediction: These methods frame conflict resolution as predicting a resolution strategy or a sequence of operations from a predefined set. RPredictor [15] uses traditional machine learning with engineered features to classify a conflict into categories. While efficient, high-level categories often still require manual intervention for complicated scenarios. MergeBERT [16] operates at a finer granularity, addressing conflicts through token-level conflict resolution classifications.

Generative and Constructive Solution Prediction: These methods aim to directly produce the resolved code. One established strategy involves *Selection and Composition*; for example, DeepMerge [29] employs Pointer Networks [30], a variation of the Attention mechanism [42], to ”generate” a resolution in a seq-to-seq way.

IX. CONCLUSION

In this work, we introduced EDITFUSION, a novel learning-based approach for automated merge conflict resolution via *edit script selection*. Our large-scale empirical study revealed the complexity and resolvability of adjacent-line conflicts, demonstrating the feasibility of conflict resolution via edit script selection. We implemented EDITFUSION, treating conflict resolution as a binary classification for each edit script. Evaluations demonstrated that EDITFUSION outperforms representative existing methods like MERGEBERT and RPREDICTOR. These findings validate the effectiveness of the edit script selection method and highlight EDITFUSION's strong potential for accurately resolving real-world merge conflicts.

ACKNOWLEDGMENT

We thank the reviewers for their constructive comments. This research was supported by NSFC (No 62272214). Any opinions, findings, and conclusion in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] C. Sung, S. K. Lahiri, M. Kaufman, P. Choudhury, and C. Wang, "Towards understanding and fixing upstream merge induced conflicts in divergent forks: An industrial case study," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '20. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 172–181.

[2] M. Mahmoudi and S. Nadi, "The android update problem: an empirical study," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 220–230. [Online]. Available: https://doi.org/10.1145/3196398.3196434

[3] G. Rooney and D. Berlin, *Practical Subversion*. Apress, 2007.

[4] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[5] C. Brindescu, M. Codoban, S. Shmarkatiuk, and D. Dig, "How do centralized and distributed version control systems impact software changes?" in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 322–333. [Online]. Available: https://doi.org/10.1145/2568225.2568322

[6] B. Shen, M. A. Gulzar, F. He, and N. Meng, "A Characterization Study of Merge Conflicts in Java Projects," *ACM Transactions on Software Engineering and Methodology*, p. 3546944, Jul. 2022.

[7] S. S. Towqir, B. Shen, M. A. Gulzar, and N. Meng, "Detecting Build Conflicts in Software Merge for Java Programs via Static Analysis," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, jan 2023, pp. 1–13.

[8] S. Apel, O. Leßenich, and C. Lengauer, "Structured merge with auto-tuning: Balancing precision and performance," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '12. New York, NY, USA: Association for Computing Machinery, sep 2012, pp. 120–129.

[9] B. Shen, W. Zhang, H. Zhao, G. Liang, Z. Jin, and Q. Wang, "IntelliMerge: A refactoring-aware software merging technique," vol. 3, pp. 1–28. [Online]. Available: https://dl.acm.org/doi/10.1145/3360596

[10] S. Apel, J. Liebig, B. Brandl, C. Lengauer, and C. Kästner, "Semistructured merge: Rethinking merge in revision control systems," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, sep 2011, pp. 190–200.

[11] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 168–178.

[12] O. Leßenich, S. Apel, and C. Lengauer, "Balancing precision and performance in structured merge," *Automated Software Engineering*, vol. 22, no. 3, pp. 367–397, 2015.

[13] T. Mens, "A state-of-the-art survey on software merging," vol. 28, no. 5, pp. 449–462.

[14] F. Zhu and F. He, "Conflict resolution for structured merge via version space algebra," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–25, 2018.

[15] W. Aldndni, N. Meng, and F. Servant, "Automatic prediction of developers' resolutions for software merge conflicts," *Journal of Systems and Software*, vol. 206, p. 111836, Dec. 2023.

[16] A. Svyatkovskiy, S. Fakhoury, N. Ghorbani, T. Mytkowicz, E. Dinella, C. Bird, J. Jang, N. Sundaresan, and S. K. Lahiri, "Program merge conflict resolution via neural transformers," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 822–833.

[17] J. Zhang, T. Mytkowicz, M. Kaufman, R. Piskac, and S. K. Lahiri, "Using pre-trained language models to resolve textual and semantic merge conflicts (experience paper)," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. Virtual South Korea: ACM, Jul. 2022, pp. 77–88.

[18] J. Dong, Q. Zhu, Z. Sun, Y. Lou, and D. Hao, "Merge conflict resolution: Classification or generation?" in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1652–1663.

[19] D. Dig, K. Manzoor, R. Johnson, and T. N. Nguyen, "Refactoring-aware configuration management for object-oriented programs," in *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 427–436.

[20] D. Dig, K. Manzoor, R. E. Johnson, and T. N. Nguyen, "Effective software merging in the presence of object-oriented refactorings," *IEEE Transactions on Software Engineering*, vol. 34, no. 3, pp. 321–335, 2008.

[21] G. Seibt, F. Heck, G. Cavalcanti, P. Borba, and S. Apel, "Leveraging Structure in Software Merge: An Empirical Study," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4590–4610, Nov. 2022.

[22] B. Shen, W. Zhang, A. Yu, Y. Shi, H. Zhao, and Z. Jin, "SoManyConflicts: Resolve Many Merge Conflicts Interactively and Systematically," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Melbourne, Australia: IEEE, Nov. 2021, pp. 1291–1295.

[23] C. Shen, W. Yang, M. Pan, and Y. Zhou, "Git Merge Conflict Resolution Leveraging Strategy Classification and LLM," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, Oct. 2023, pp. 228–239.

[24] Q. Zhang, L. Su, K. Ye, and C. Qian, "CONGRA: Benchmarking Automatic Conflict Resolution," Sep. 2024.

[25] H. Le Nguyen and C.-L. Ignat, "Parallelism and conflicting changes in git version control systems," in *IWCES'17-The Fifteenth International Workshop on Collaborative Editing Systems*, 2017.

[26] G. Ghiotto, L. Murta, M. Barros, and A. van der Hoek, "On the Nature of Merge Conflicts: A Study of 2,731 Open Source Java Projects Hosted by GitHub," *IEEE Transactions on Software Engineering*, vol. 46, no. 8, pp. 892–915, aug 2020.

[27] P. Elias, H. d. S. C. Junior, E. Ogasawara, and L. G. P. Murta, "Towards Accurate Recommendations of Merge Conflicts Resolution Strategies," Rochester, NY, Jan. 2023.

[28] P. Yin, G. Neubig, M. Allamanis, M. Brockschmidt, and A. L. Gaunt, "Learning to Represent Edits," feb 2019.

[29] E. Dinella, T. Mytkowicz, A. Svyatkovskiy, C. Bird, M. Naik, and S. K. Lahiri, "DeepMerge: Learning to Merge Programs," sep 2021.

[30] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," in *Advances in Neural Information Processing Systems*, vol. 28. Curran Associates, Inc., 2015.

[31] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[32] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[33] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[34] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1536–1547. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.139

[35] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[36] G. Team, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," 2024. [Online]. Available: https://arxiv.org/abs/2403.05530

[37] O. Leßenich, S. Apel, C. Kästner, G. Seibt, and J. Siegmund, "Renaming and shifted code in structured merging: Looking ahead for precision and performance," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 543–553.

[38] S. Böcker, D. Bryant, A. W. Dress, and M. A. Steel, "Algorithmic aspects of tree amalgamation," *Journal of Algorithms*, vol. 37, no. 2, pp. 522–537, 2000.

[39] K. Zhang and T. Jiang, "Some max snp-hard results concerning unordered labeled trees," *Information Processing Letters*, vol. 49, no. 5, pp. 249–254, 1994.

[40] A. Trindade Tavares, P. Borba, G. Cavalcanti, and S. Soares, "Semistructured merge in javascript systems," in *2019 34th IEEE/ACM Interna-*

tional Conference on Automated Software Engineering (ASE), 2019, pp. 1014–1025.

[41] G. Cavalcanti, P. Borba, G. Seibt, and S. Apel, "The impact of structure on software merging: Semistructured versus structured merge," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1002–1013.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.