Graph Enhanced Transformer for Semi-Supervised Duplicate Bug Report Detection

Kerem Bayramoğlu*
kerem.bayramoglu@bilkent.edu.tr
Bilkent University
Dept. of Computer Engineering
Çankaya, Ankara, Türkiye

Hüseyin Karaca
huseyin.karaca@bilkent.edu.tr
Bilkent University
Dept. of Electrical and Electronics Engineering
Çankaya, Ankara, Türkiye

Emirhan Koç
emirhan.koc@bilkent.edu.tr
Bilkent University
Dept. of Electrical and Electronics Engineering
Çankaya, Ankara, Türkiye

Hasan Erkin Ünlü
erkin.unlu@bilkent.edu.tr
Bilkent University
Dept. of Electrical and Electronics Engineering
Çankaya, Ankara, Türkiye

Rabia Ela Ünlü
ela.unlu@bilkent.edu.tr
Bilkent University
Dept. of Electrical and Electronics Engineering
Çankaya, Ankara, Türkiye

ABSTRACT

Context. Duplicate bug report detection aims to identify bug reports that describe the same underlying problem or can be resolved by the same fix. In large-scale repositories, duplicates increase triage workload and waste developer time. While transformer-based sentence encoders improve semantic matching compared to classical IR baselines, they rely on labeled duplicate pairs, which are typically scarce in real-world repositories. Moreover, exhaustive pairwise training is computationally infeasible and cannot effectively incorporate the full pool of unlabeled reports.

Method. We propose a semi-supervised Graph-Enhanced Transformer framework that couples transformer-based semantic encoding with graph-based message passing. We represent the full corpus (train/validation/test) as nodes in a title-similarity graph built from `bert-base-uncased` title embeddings; to obtain a more informative similarity geometry, we apply PCA before computing similarities (thresholding can be used as an optional sparsification step). During training, transformer `[CLS]` embeddings are projected to a compact space and injected into a two-layer Graph Convolutional Network (GCN) , enabling information propagation from supervised nodes to the unlabeled portion of the corpus. Supervision is applied only on a feasible subset of labeled positive pairs and sampled negatives via a cosine embedding objective, while the remaining unlabeled reports contribute indirectly through graph propagation. For evaluation, model parameters are frozen and pairwise predictions are obtained by thresholding cosine similarity, where the decision threshold is selected on the validation split.

Results. Experiments on Eclipse Platform and Mozilla Thunderbird show that graph propagation enables the model to benefit from unlabeled reports under label-scarce conditions, yielding stable training dynamics and competitive classification performance relative to transformer-only baselines. Source code and data are publicly available at the project repository.[1]

Conclusions. The proposed graph-enhanced training strategy offers a principled way to exploit unlabeled bug reports without requiring explicit pairwise supervision for transformers. By confining graph-based reasoning to the transductive training phase, the framework balances label efficiency, scalability, and practical deployment constraints. These results indicate that graph-structured semi-supervised learning can effectively complement transformer-based encoders for duplicate bug report detection in realistic, label-scarce settings.

*Authors are listed in alphabetical order by surname.

KEYWORDS

Bug Report, Bug Duplicate Detection, Transformers, BERT, LLMs, Graph Neural Networks, Semi-Supervised Learning

ACM Reference Format:
Kerem Bayramoğlu, Hüseyin Karaca, Emirhan Koç, Hasan Erkin Ünlü, and Rabia Ela Ünlü. 2018. Graph Enhanced Transformer for Semi-Supervised Duplicate Bug Report Detection. In *Proceedings of Data Science For Software Engineering (CS 588).* ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In large software developments, bug-tracking systems are used as a tool for handling software bug reports, as well as handling software feature requests. In the course of time, as software bug repositories databases continue to expand, it has been noticed that there exist overlapping software bug reports from diverse users. This results in the enlargement of the size of the software bug-reporting databases, as well as consuming the time of software developers in examining the same software bugs, as these overlapping software bug reports are nothing but duplicates. Studies show that a large number of

[1]https://github.com/huseyin-karaca/graph-enhanced-dbd.git

duplicate bug reports exist in repositories; for example, 20% of reports in Eclipse and 30% in Firefox were marked as duplicate [2].

Automating duplicate bug report detection has therefore become an essential research direction in software engineering. Classical information retrieval (IR) techniques such as Term Frequency–Inverse Document Frequency (TF-IDF), Best Matching 25 with field weighting (BM25F), and custom retrieval functions (e.g., REP) have been widely used to match new bug reports with existing ones [32].

Although these algorithms are computationally efficient, they remain limited in addressing the vocabulary mismatch problem, where semantically similar bug reports are expressed using different lexical choices [8]. More recent approaches based on deep learning, particularly transformer-based models derived from BERT, have demonstrated improved effectiveness in capturing semantic similarity by leveraging contextual representations of bug report titles and descriptions [5, 20, 27]. Despite these advances, transformer-based methods rely heavily on supervised learning with labeled duplicate reports, which introduces a substantial dependency on annotated data. In practice, acquiring such labels is costly and labor-intensive, and real-world bug repositories typically contain only a limited number of confirmed duplicate annotations. To mitigate the reliance on limited annotated data, several studies have proposed augmenting supervision by exploiting unlabeled bug reports, for instance through pairing strategies combined with negative sampling [22, 26]. While such approaches partially alleviate label scarcity, they remain limited in their ability to fully leverage the entire pool of available unlabeled data, and thus do not provide a comprehensive solution for large-scale, label-sparse bug repositories.

To address these limitations, we propose a semi-supervised Graph-Enhanced Transformer framework for duplicate bug report detection. Our approach leverages the semantic power of transformer encoders, along with the relational reasoning power of graph neural networks (GNNs). Unlike transformer-based models, which are inherently limited in their ability to fully exploit unlabeled data, our framework explicitly incorporates both labeled and unlabeled bug reports as nodes within a unified graph structure. By leveraging the message-passing mechanism of GNNs, information is propagated between labeled and unlabeled nodes, enabling implicit knowledge transfer across the entire report collection. Although the training objective and final predictions are defined solely over labeled data, the inclusion of unlabeled nodes allows the model to benefit from their structural and semantic context, preventing their complete exclusion from the learning process.

This study is guided by the following research questions:

- **RQ1:** Can unlabeled bug reports be effectively leveraged through graph-based representations for duplicate bug report detection?
- **RQ2:** Can a scalable architecture be designed to support duplicate bug report detection in large-scale bug repositories?
- **RQ3:** Does the proposed graph-enhanced, semi-supervised framework achieve competitive performance compared to strong transformer-based baselines?

This study offers the following contributions:

- We explicitly model both **labeled and unlabeled bug reports as nodes in a unified graph**, enabling effective exploitation of unlabeled data through graph-based message passing, while defining supervision and prediction solely on labeled samples.
- We show that unlabeled bug reports contribute indirectly to learning by providing **structural and semantic context**, facilitating implicit knowledge transfer across the entire report collection.
- To the best of our knowledge, this work represents **an early exploration of GNN-based approaches for duplicate bug report detection**, revealing the potential of graph-enhanced learning in this domain.
- Although the proposed framework adopts a **transductive training and inference setting**, it is **not inherently restricted to this regime** and can be naturally extended to **inductive inference** with appropriate architectural modifications.

2 PROBLEM DESCRIPTION AND MOTIVATION

2.1 Formal Problem Definition

The core challenge in Duplicate Bug Report Detection (DBRD) is to automatically identify whether two bug reports describe the same underlying software defect. Formally, given a bug repository $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ where each report r_i contains textual fields (title, description) and metadata (timestamp, reporter, component), the task is to learn a function $f : \mathcal{R} \times \mathcal{R} \rightarrow \{0, 1\}$ that predicts whether a pair of reports (r_i, r_j) are duplicates. This is inherently difficult due to lexical variations, incomplete descriptions, domain-specific terminology, and reporting style differences across users.

In realistic software development environments, the availability of labeled data is severely limited. While a repository may contain tens of thousands of bug reports, only a small fraction (typically less than 5%) have confirmed duplicate relationships manually identified by developers or triagers. The remaining reports remain unlabeled, representing a vast untapped resource that conventional supervised learning approaches cannot effectively utilize.

2.2 Limitations of Existing LLM-Based Approaches

Transformer-based language models such as BERT, RoBERTa, and their domain-adapted variants have demonstrated strong performance in semantic text matching tasks. However, these models face fundamental limitations in the DBRD setting. Consider a concrete scenario: a repository contains 20,000 bug reports with only 500 confirmed duplicate pairs (1,000 labeled reports). A pairwise BERT-based classifier would be trained on these 500 positive pairs plus an equal number of sampled negative pairs, effectively utilizing only 1,000 reports while completely disregarding the remaining 19,000 unlabeled reports.

The quadratic growth of potential pairs exacerbates this problem. For n bug reports, there exist $\binom{n}{2} = \frac{n(n-1)}{2}$ possible pairs. For 20,000 reports, this yields approximately 200 million pairs, making exhaustive pairwise training computationally infeasible. Existing approaches employ negative sampling strategies, but these methods lack a principled mechanism to propagate semantic information across the unlabeled corpus. Moreover, they are limited by the explicit supervision paradigm: without a label for a pair (r_i, r_j), the model receives no training signal involving those reports together.

2.3 Illustrative Example: LLM Failure Under Data Scarcity

To illustrate the practical limitation, consider two bug reports from an Eclipse repository: *Report A (ID 12345):* "NullPointerException in editor when opening XML file with invalid schema reference" *Report B (ID 67890):* "NPE thrown during XML editor initialization with broken schema link." These reports clearly describe the same bug using different terminology ("NullPointerException" vs. "NPE", "opening" vs. "initialization", "invalid reference" vs. "broken link"). A well-trained transformer model with sufficient labeled examples from the XML editor domain would likely identify this pair as duplicates through semantic similarity. However, suppose the training set contains only 100 labeled duplicate pairs, none of which involve the XML editor component or schema-related issues. The transformer model, trained exclusively on these 100 pairs, would lack the domain-specific context to recognize the semantic equivalence between Reports A and B. Meanwhile, the repository contains 500 other unlabeled reports related to XML editing, including variations of similar terminology and error patterns. A purely supervised LLM-based approach cannot leverage these 500 reports during training, as they lack explicit duplicate labels.

2.4 Motivation for Graph-Based Semi-Supervised Learning

Graph neural networks provide a natural solution to overcome these limitations. Unlike pairwise supervised learning, GNNs operate on relational neighborhoods and propagate information through message passing across connected nodes without requiring explicit labels for every connection. This enables us to construct a unified graph representation where all available bug reports—both labeled and unlabeled—participate in the learning process.

In our framework, edges are formed based on label-independent semantic similarity (e.g., cosine similarity of title embeddings), enabling efficient graph construction without additional manual annotations. The transformer component operates on a restricted but feasible subset of report pairs during training: positive pairs from known duplicates and negative pairs sampled from different groups. The representations learned from these labeled pairs are embedded as graph node features, and the GNN propagates this information across the entire graph through neighborhood aggregation.

Returning to the XML editor example: even though Reports A and B may not appear in the training set, if they are connected through intermediate nodes (other XML-related reports) in the graph, the GNN can propagate relevant semantic information to them. Although direct supervision is applied only to nodes in labeled pairs, the graph structure allows information flow to unpaired and unlabeled nodes, enabling the model to benefit from the full corpus. This design decouples pairwise supervision from global data utilization, making it suitable for realistic, label-scarce duplicate bug report detection scenarios.

3 RELATED WORK

The problem of duplicate bug report detection has been a central challenge in software engineering research for many years. Early approaches primarily relied on classical Information Retrieval (IR) methods, while more recent techniques have leveraged advances in machine learning and deep learning to improve performance. In this section, we review the evolution of methodologies for DBRD, highlighting key contributions and their limitations.

3.1 Information Retrieval Models

Early automated approaches framed DBRD as a classical Information Retrieval (IR) problem [33], typically using the Vector Space Model (VSM) [23]. In this model, each bug report is treated as a document and is transformed into a high-dimensional vector. The components of this vector are weighted based on the terms present in the document.

The most common scheme is Term Frequency-Inverse Document Frequency (TF-IDF) [29, 31]. This method assigns high weights to terms that are frequent in a specific document but rare across the entire corpus. Once vectorized, report similarity is calculated using Cosine Similarity [23].

The fundamental limitation of this paradigm is the “vocabulary mismatch problem" [9]. First identified by Furnas et al. [9], this refers to the low probability that two people will use the same terms for the same concept.

3.2 Machine Learning Approaches

In response, researchers applied supervised machine learning, shifting the problem from retrieval to classification (a binary duplicate/non-duplicate decision) [12, 33]. Early work, such as Jalbert and Weimer (2008) [12] and Sun et al. (2010) [33], applied classifiers like Support Vector Machines (SVMs) to feature pairs.

However, training a classifier requires generating $O(N^2)$ potential pairs, where N denotes the total number of bug reports in the repository , which is computationally intractable [33]. This "low efficiency" of pair-wise classification led to a critical development: using machine learning to improve the IR model itself.

This led to REP, a retrieval function proposed by Sun et al. [32] that became a dominant baseline. REP extends the Okapi BM25F formula to create a custom similarity function. Its key innovation is combining textual similarity with structured metadata (product, component, version) [32]. It then uses supervised learning to learn the optimal weights for each field, tuning its function for a specific bug repository [32]. The success of REP demonstrated that a specialized, feature-aware IR model could outperform general-purpose ML models in both accuracy and efficiency [32, 33], and it remained a difficult baseline to beat for years [32].

3.3 Deep Learning for Semantic Representation

The deep learning (DL) revolution promised to finally solve the vocabulary mismatch problem by learning true semantic meaning [3]. Initial attempts with Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), such as the Dual-Channel CNN (DCCNN) by He et al. (2020) [11], began learning vector embeddings from report text. However, these early DL models often struggled to outperform the highly-optimized REP baseline [32].

A significant shift occurred with the Transformer model, specifically BERT [6]. BERT's pre-training allows it to capture deep, contextual understanding of language [6]. This base model was quickly followed by optimized variants such as RoBERTa [21], which improved performance by refining the pre-training process, and ALBERT [19], which focused on parameter reduction. However, applying these cross-encoder models (BERT, RoBERTa, ALBERT) directly to DBRD exposed a critical flaw. As a cross-encoder, BERT requires both reports to be fed into the network simultaneously [28]. Finding the best match for a new report in a large database would require 10,000s of computations, a process estimated to take around 65 hours [28], rendering it unusable.

This was solved by Reimers and Gurevych (2019) with Sentence-BERT (SBERT) [28]. SBERT adapts BERT using a siamese network structure [28]. Two identical BERT models process each bug report independently, producing a single "sentence embedding" for the pair [28]. Because embeddings are generated independently, they can be pre-computed. A new report can be embedded once, and its similarity to all others found almost instantaneously via cosine similarity search [28]. This reduces the 65-hour search task to approximately 5 seconds [28]. SBERT provides a powerful, semantic-native replacement for TF-IDF, solidifying the dominance of the two-stage "retrieval-rerank" pipeline. A recent comparative study by Meng et al. (2024) empirically evaluated many of these architectures, confirming the performance gains of transformer-based models like BERT, ALBERT, and RoBERTa over earlier neural architectures like Bi-LSTM and DC-CNN [24].

3.4 Hybrid Systems and Emerging Approaches

Current state-of-the-art systems are often hybrid approaches combining lexical precision with semantic understanding [25]. The DBTM (Duplicate Bug report Topic Model) approach, for example, combines the IR model BM25F with topic-based features [25], allowing it to match reports based on higher-level "technical bugs" [25].

More recently, this philosophy has extended to Large Language Models (LLMs). Cupid [34] achieves state-of-the-art results by combining an LLM (ChatGPT) with the classic IR model REP [32, 34]. The LLM is used in a zero-shot setting as a semantic pre-processor to "get essential information" from the raw bug report [34]. This "cleaned" information is then fed to the domain-aware REP model for the final similarity calculation. This hybrid approach was shown to improve recall over previous baselines by a significant margin [34].

While the aforementioned models have progressively advanced semantic understanding, they remain almost exclusively *supervised*, requiring large, costly datasets of labeled duplicate pairs. As noted in recent studies, transformer models like BERT and its variants (ALBERT, RoBERTa) excel when data is plentiful, but their performance is limited in the more realistic, label-scarce environments common to bug repositories [24]. Furthermore, as GNNs emerge as a new frontier, existing applications are often *transductive*, meaning they cannot perform inference on new, unseen bug reports without retraining [10].

Our proposed framework addresses these two specific, critical gaps: (1) label scarcity and (2) inference scalability. We operate in a *semi-supervised* setting, using a GNN to leverage the vast majority of *unlabeled* reports during training. Table 1 outlines this positioning relative to closely related works.

Stepping back, we view DBRD as both a semantic and a relational problem. In the next section, we model the bug repository as a graph and use an inductive GNN (e.g., GraphSAGE) to produce embeddings for unseen reports without retraining [10]. To deal with limited supervision and sharpen the decision boundary, we add two simple training aids: generative augmentation to create within-class variants [1] and hard-negative mining to focus the model on near-miss non-duplicates. We detail the architecture and training strategy in the next section.

4 PROPOSED METHOD

We introduce a semi-supervised GNN augmented transformer-encoder framework for identification of duplicate bug reports, where the model is trained primarily in positive-pair supervision with a regularization strategy, i.e., the negative pair sampling . Overall, the pipeline consists of three main stages: (i) Data Processing, where duplicate labels are transformed into positive training pairs and all remaining pairs are treated as candidate negatives; (ii) Embedding Construction, Training as well as Validation, where a transformer encoder learns similarity-preserving representations using a binary similarity objective; and (iii) Inference, where the model is tested on untrained positive and negative pairs. The overall schematic encompassing data processing, embedding construction, information transfer to the GNN and training is illustrated in Fig. 1. The description and formal definition of each stage is provided as follows:

Pair Construction. Let $\mathcal{P} = \{p_1, p_2, \dots, p_N\}$ denote the set of all bug reports, where each report p_i is associated with a duplicate-group label $g_i \in \{1, \dots, L\}$.

Positive pairs. For each report index $i \in \{1, 2, \dots, N\}$, we define the set of positive (duplicate) pairs as

$$\mathcal{P}_i^{(+)} = \{(i, j) \mid g_i = g_j,\ \ j \neq i\}.$$

These pairs enforce similarity among reports belonging to the same duplicate group.

Anchor-based negative pairs. For each report index $i \in \{1, 2, \dots, N\}$, we define the anchor-based negative pair set as

$$\mathcal{N}_i^{(a)} = \{(i, j) \mid g_i \neq g_j\}.$$

Since $|\mathcal{N}_i^{(a)}|$ is typically much larger than $|\mathcal{P}_i^{(+)}|$, we randomly sample a subset

$$\tilde{\mathcal{N}}_i^{(a)} \subset \mathcal{N}_i^{(a)}, \qquad |\tilde{\mathcal{N}}_i^{(a)}| = \alpha\, |\mathcal{N}_i^{(a)}|,$$

where α denotes the fixed subsampling ratio.

Cross-group random negative pairs. To further diversify negative supervision, we additionally construct anchor-free negative pairs by randomly sampling report pairs across different duplicate groups:

$$\mathcal{N}^{(r)} = \{(i, j) \mid i \neq j,\ g_i \neq g_j\}.$$

From this set, a random subset $\tilde{\mathcal{N}}^{(r)} \subset \mathcal{N}^{(r)}$ is selected.

Table 1: Comparison of Closely Related DBRD Methodologies

Methodology	Core Technology	Semantic Capacity	Uses Unlabeled Data?	Inference Scalability
Bi-LSTM [4]	Recurrent Neural Network (RNN)	Sequential	No (Supervised)	High (Bi-encoder)
DC-CNN [11]	Convolutional Neural Network (CNN)	Local Patterns	No (Supervised)	High (Bi-encoder)
BERT/ALBERT/RoBERTa [6, 19, 21]	Transformer (Cross-Encoder)	Deep Contextual	No (Supervised)	Low (Pair-wise)
SBERT [28]	Transformer (Siamese Bi-Encoder)	Deep Contextual	No (Supervised)	High (Bi-encoder)
Our Method	**Transformer + GNN (Hybrid)**	**Deep Contextual & Relational**	**Yes (Semi-supervised)**	**High (Bi-encoder)**

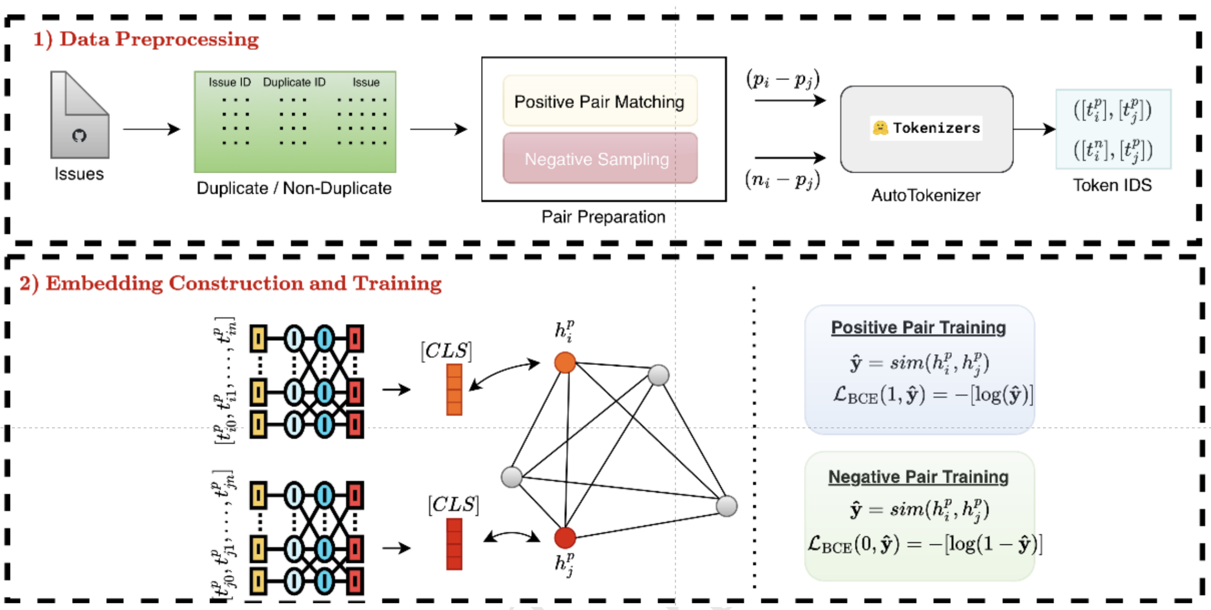
1) Data Preprocessing
Issues
Issue ID
Duplicate ID
Issue
Duplicate / Non-Duplicate
Positive Pair Matching
Negative Sampling
Pair Preparation
$(p_i - p_j)$
$(n_i - p_j)$
Tokenizers
AutoTokenizer
$([t_i^p], [t_j^p])$
$([t_i^n], [t_j^p])$
Token IDS
2) Embedding Construction and Training
$[t_{i0}^p, t_{i1}^p, \dots, t_{in}^p]$
$[t_{j0}^p, t_{j1}^p, \dots, t_{jm}^p]$
$[CLS]$
$[CLS]$
h_i^p
h_j^p
Positive Pair Training
$\hat{\mathbf{y}} = sim(h_i^p, h_j^p)$
$\mathcal{L}_{\mathrm{BCE}}(1, \hat{\mathbf{y}}) = -[\log(\hat{\mathbf{y}})]$
Negative Pair Training
$\hat{\mathbf{y}} = sim(h_i^p, h_j^p)$
$\mathcal{L}_{\mathrm{BCE}}(0, \hat{\mathbf{y}}) = -[\log(1 - \hat{\mathbf{y}})]$

Figure 1: Overall pipeline of the proposed method. Upper: The bugs are documented with bug numbers, titles, and descriptions. The positive and negative pairs are collected and stacked. Then, the token IDs are listed to be fed into BERT. Bottom: The graph is constructed, then BERT is employed for feature extractor and these embeddings transferred to the GNN component. Finally, the model is trained based on similarity and dissimilarities between pairs.

Final training pair sets. The final positive and negative pair sets used during training are

$$\mathcal{P}^{(+)} = \bigcup_{i=1}^{N} \mathcal{P}_i^{(+)}, \qquad \mathcal{N} = \bigcup_{i=1}^{N} \tilde{\mathcal{N}}_i^{(a)} \cup \tilde{\mathcal{N}}^{(r)}.$$

Following pair construction, each bug report is converted into a token-level representation using a HuggingFace tokenizer. Let x_i denote the textual content of report p_i, obtained by concatenating its title and description.

For each positive pair $(i, j) \in \mathcal{P}_i^{(+)}$ and each sampled negative pair $(i, j) \in \tilde{\mathcal{N}}_i^{(a)} \cup \tilde{\mathcal{N}}^{(r)}$, we obtain the corresponding token-ID sequences as

$$t_i = \text{Tokenizer}(x_i), \qquad t_j = \text{Tokenizer}(x_j).$$

The same tokenization process is applied to both positive and negative pairs, ensuring a unified input format for the transformer encoder. Each token-ID sequence is then used as direct input to the transformer-based embedding module, which is described in the following sections.

4.1 Embedding and Graph Construction

In the embedding construction stage, our methodology consists of three fundamental components: *Graph Construction*, *Embedding Construction*, and *End-to-End Training*.

4.1.1 *Graph Construction.*

GNNs provide a flexible representation learning framework capable of operating on both labeled and unlabeled nodes. A GNN defines a message-passing operator over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where information is iteratively exchanged among neighboring nodes to produce context-aware node embeddings [13, 14]. Since the update functions in a GNN do not rely on node labels, the model naturally supports semi-supervised settings in which only a subset of nodes is labeled while the remaining nodes remain unlabeled [16]. This property is particularly well aligned with our setting, where duplicate-group information is available for only a small fraction of reports.

We construct the graph using the entire dataset, such that every bug in the corpus, including training, validation, and test samples, is represented as a node. Each node corresponds to a unique bug report, and node indices are explicitly preserved to ensure a consistent mapping between reports and their graph representations throughout training and inference. Although GNNs operate in a permutation-invariant manner, maintaining this index correspondence is essential for correctly associating learned representations with their respective reports. Edges in the graph encode pairwise semantic relations between bugs; while each edge can be interpreted as a potential report pair, direct connectivity between all relevant nodes is not required, as information can propagate across the graph through multi-hop message passing.

In our graph construction, each bug is assigned an initial node feature vector corresponding to a one-hot encoding of its node index. Formally, for a graph with N nodes, the initial embedding of node i is given by

$$x_i^{(0)} = e_i \in \mathbb{R}^N,$$

where e_i denotes the i-th standard basis vector.

The edge set is constructed using semantic similarity computed from bug titles. Let τ_i denote the tokenized title of bug i. Each title is first encoded using a pretrained `bert-base-uncased` model, producing a contextual embedding in $\mathbb{R}^{768}$. Based on empirical observations, similarity computations performed directly in this high-dimensional space tend to be overly smooth and less informative for graph construction. To mitigate this effect, we apply Principal Component Analysis (PCA) to the title embeddings and retain the top $d = 10$ principal components, resulting in reduced representations $\tilde{\tau}_i \in \mathbb{R}^{10}$. Similarity is then computed in this reduced space, yielding more discriminative and semantically meaningful relationships by alleviating redundancy and noise in the original embeddings.

Edges are added between nodes based on the resulting similarity structure, without enforcing a hard threshold. Instead, similarity values are used directly to define graph connectivity, enabling flexible and dense relational modeling across the dataset. While threshold-based edge selection constitutes a viable alternative for graph sparsification, our formulation does not rely on an explicit similarity cutoff.

4.1.2 Embedding Construction. For each token-ID sequence obtained in the data preprocessing stage, the embedding construction module feeds the sequence into a pre-trained transformer encoder. Each element of a report pair is processed independently through the encoder, yielding two contextualized representations. In practice, this corresponds to two parallel forward passes through a pair of parameter-tied transformer encoders, ensuring that both inputs are mapped into a shared representation space. These operations are applied uniformly to both positive and negative pair elements.

Formally, given a token-ID sequence

$$t^{(i)} = [t_0^{(i)}, \dots, t_{L_i}^{(i)}],$$

the pretrained transformer encoder is denoted by

$$f_{\mathbf{W}} : \mathbb{N}^{L_i} \to \mathbb{R}^{L_i \times d},$$

where $\mathbf{W}$ represents the shared parameters of the weight-tied encoders and d denotes the hidden dimension of the transformer. The resulting contextual embedding matrix is

$$H_i = f_{\mathbf{W}}(t^{(i)}) \in \mathbb{R}^{L_i \times d}.$$

The report-level embedding is extracted from the special `[CLS]` token position,

$$h_i^{(\mathrm{CLS})} = H_i[0] \in \mathbb{R}^d,$$

which serves as a pooled summary representation of the entire input sequence [16, 17].

To reduce computational and memory overhead, particularly under small-scale computing constraints, the high-dimensional `[CLS]` embeddings are further projected into a lower-dimensional space using a learnable linear transformation:

$$z_i = \mathbf{W}_p\, h_i^{(\mathrm{CLS})} + \mathbf{b}_p, \qquad \mathbf{W}_p \in \mathbb{R}^{D \times d},$$

yielding compact embeddings $z_i \in \mathbb{R}^D$. In our experiments, D is treated as a tunable hyperparameter, with $D = 128$ used as a stable operating point unless otherwise stated.

For each supervised pair $(i, j) \in \mathcal{S}$, the final embeddings are obtained via two parallel forward passes followed by projection:

$$z_i = \mathbf{W}_p\, f_{\mathbf{W}}(t^{(i)})[0], \qquad z_j = \mathbf{W}_p\, f_{\mathbf{W}}(t^{(j)})[0].$$

Parameter sharing across both the transformer encoder and the projection layer ensures that all reports are embedded into a consistent low-dimensional representation space suitable for efficient similarity-based training and inference.

4.1.3 Training Pipeline. In the training pipeline, we combine the outputs of the transformer-based embedding module and the graph neural network. The token-ID sequences obtained from the report descriptions are first fed into the pretrained transformer encoder, producing the corresponding `[CLS]` embeddings. For a report pair (i, j), let h_i and h_j denote the transformer-derived `[CLS]` embeddings. These embeddings are then injected into the GNN by assigning them to their corresponding nodes in the constructed graph, ensuring consistency with the fixed node ordering. Importantly, supervision is applied only to nodes whose indices belong to the training split, while validation and test nodes are included in the graph to enable information propagation via message passing.

Fusion of Transformer and GNN representations. Let $z_i^{(\mathrm{tr})} \in \mathbb{R}^D$ denote the projected transformer embedding of report i, and let $z_i^{(\mathrm{gnn})} \in \mathbb{R}^D$ denote the corresponding GNN output embedding after message passing. The final embedding used for optimization is obtained via a weighted fusion:

$$z_i = \lambda\, z_i^{(\mathrm{tr})} + (1 - \lambda)\, z_i^{(\mathrm{gnn})}, \qquad \lambda \in [0, 1].$$

This formulation balances semantic information captured by the transformer with relational information captured by the GNN. As an alternative design choice, the two embeddings may also be combined via concatenation, i.e., $z_i = [z_i^{(\mathrm{tr})}; z_i^{(\mathrm{gnn})}]$, followed by a projection layer. Unless otherwise stated, the weighted-sum fusion is used throughout this work.

Cosine embedding objective. For each supervised report pair (i, j), we compute the cosine similarity

$$s_{ij} = \cos(z_i, z_j).$$

We optimize a cosine embedding loss, where pairwise labels are mapped to $\{-1, +1\}$. Specifically, positive (duplicate) pairs are assigned $y_{ij} = +1$, while sampled negative pairs are assigned $y_{ij} = -1$. The loss for a pair (i, j) is defined as

$$\mathcal{L}_{ij} = \begin{cases} 1 - s_{ij}, & y_{ij} = +1, \\ \max(0,\ s_{ij} - m), & y_{ij} = -1, \end{cases}$$

where $m \in [0, 1]$ denotes a margin hyperparameter. The overall training objective is given by

$$\mathcal{L} = \sum_{(i,j)\in\mathcal{P}^{(+)}} \mathcal{L}_{ij} + \sum_{(i,j)\in\mathcal{N}} \mathcal{L}_{ij}.$$

Minimizing $\mathcal{L}$ jointly updates the parameters of the transformer encoder, the projection module, and the GNN, enabling similarity-preserving representations informed by both textual semantics and graph-structured relational information.

Validation-based decision rule. During validation, similarity scores are computed only for pairs involving nodes belonging to the validation split. A pair (i, j) is predicted as duplicate if its similarity score exceeds a threshold θ:

$$\hat{y}_{ij} = \mathbb{I}\big[s_{ij} \geq \theta\big].$$

A default threshold of $\theta = 0.5$ is used as an initial operating point during training. To determine an appropriate decision threshold, we perform a validation-time grid search over $\theta \in [0, 1]$ with a step size of 0.05, and select the value that yields the best validation performance. The selected threshold is then fixed for subsequent evaluation.

4.2 Inference Procedure

Let $\hat{f}_{\mathrm{W}}$ and $\hat{g}_{\Theta}$ denote the transformer encoder (including projection) and the GNN with parameters fixed after training. During inference, all model parameters are frozen and no further updates are performed.

Pairwise Inference on Test Nodes. For test nodes indexed by $\mathcal{I}_{\text{test}}$, we follow a procedure analogous to validation. For each test pair (i, j) with $i, j \in \mathcal{I}_{\text{test}}$, embeddings are computed as

$$z_i = \hat{f}_{\mathrm{W}}(t^{(i)}), \qquad z_j = \hat{f}_{\mathrm{W}}(t^{(j)}),$$

optionally refined via graph propagation using $\hat{g}_{\Theta}$. The cosine similarity

$$s_{ij} = \cos(z_i, z_j)$$

is then compared against the decision threshold θ, yielding the prediction

$$\hat{y}_{ij} = \mathbb{I}\big[s_{ij} \geq \theta\big].$$

This pairwise evaluation protocol is adopted to ensure consistency with existing benchmarks and prior work.

5 EXPERIMENTS AND RESULTS

5.1 Dataset

Our experimental evaluation employs two prevalently utilized benchmark datasets for duplicate bug report detection. Eclipse Platform, and Mozilla Thunderbird [18]. These include both original and duplicate reports from large, long duration software projects. It can be referred to Table 2 for details.

5.2 Experimental Setup

5.2.1 Evaluation Metrics. Let $\mathcal{T} \subseteq \mathcal{I}_{\text{test}} \times \mathcal{I}_{\text{test}}$ denote the set of evaluated test pairs, and let $y_{ij} \in \{0, 1\}$ and $\hat{y}_{ij} \in \{0, 1\}$ denote the ground-truth and predicted duplicate labels for a pair $(i, j) \in \mathcal{T}$, respectively. We define the sets of true positives, false positives, true negatives, and false negatives as

$$\begin{aligned} \mathrm{TP} &= \{(i,j) \in \mathcal{T} \mid y_{ij} = 1 \wedge \hat{y}_{ij} = 1\}, \\ \mathrm{FP} &= \{(i,j) \in \mathcal{T} \mid y_{ij} = 0 \wedge \hat{y}_{ij} = 1\}, \\ \mathrm{TN} &= \{(i,j) \in \mathcal{T} \mid y_{ij} = 0 \wedge \hat{y}_{ij} = 0\}, \\ \mathrm{FN} &= \{(i,j) \in \mathcal{T} \mid y_{ij} = 1 \wedge \hat{y}_{ij} = 0\}. \end{aligned}$$

Using these quantities, accuracy is computed as

$$\text{Accuracy} = \frac{|\mathrm{TP}| + |\mathrm{TN}|}{|\mathrm{TP}| + |\mathrm{TN}| + |\mathrm{FP}| + |\mathrm{FN}|}.$$

Precision and recall are defined as

$$\text{Precision} = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FP}|}, \qquad \text{Recall} = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FN}|},$$

and the F1-score is given by

$$\mathrm{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

5.2.2 Implementation Details. Our implementation is built using PyTorch and the HuggingFace Transformers library. For the transformer encoder, we experiment with four pre-trained models: BERT [5], RoBERTa [21], DistilBERT [30], and CodeBERT [7]. The graph neural network component employs a two-layer Graph Convolutional Network (GCN) architecture [15].

For training, we use the AdamW optimizer with a learning rate of 2×10^{-5} and a batch size of 64. Models are trained for up to 5 epochs with early stopping based on validation performance; specifically, training is stopped if validation loss does not improve for 3 consecutive epochs (patience=3) . Negative sampling is performed using two complementary strategies. First, anchor-based negative pairs are sampled for each report in proportions comparable to the number of positive pairs. Second, to increase diversity, an additional set of 5,000 negative pairs is randomly sampled across different duplicate groups. Importantly, the vast majority of unlabeled reports are not included in transformer-based pairwise training; instead, they are incorporated exclusively as nodes in the graph, enabling them to influence learning through GNN-based message passing without being explicitly paired.

Graph construction is performed using semantic similarity computed from bug titles encoded with a pretrained `bert-base-uncased` model. Similarity is computed directly on these title embeddings after PCA-based dimensionality reduction, and no explicit threshold is applied for edge selection. This design choice allows dense

Table 2: Statistics of the datasets used for duplicate bug report detection.

Metric	Eclipse	Thunderbird
Total	68, 124	26, 040
Non-Dup.	50, 606	15, 994
Dup.	17, 512	10, 046
% Dup.	25.7	38.6
Clusters	6, 282	2, 772
Pairs	87, 224	53, 156

and flexible connectivity in the graph while avoiding sensitivity to threshold tuning.

All experiments were conducted on a single NVIDIA high performance A100 GPU with 80 GB VRAM. All source code and the full reproducibility package, including all necessary data files and scripts, can be found in the huseyin-karaca/graph-enhanced-dbd GitHub repository.

5.3 Results

We organize our experimental results around the three research questions presented in Section 1.

5.3.1 Leveraging Unlabeled Data Through Graph Structure (RQ1). To investigate whether unlabeled bug reports can be effectively leveraged through graph-based representations, we analyze the similarity distributions in the learned embedding space. Table 3 presents representative examples from the validation set, showing the cosine similarity scores for both duplicate and non-duplicate pairs.

Table 3: Representative similarity scores on validation set pairs. Duplicate pairs consistently achieve high similarity, while non-duplicate pairs show low or negative similarity.

Duplicates (Label = 1)			Non-Duplicates (Label = 0)		
Issue 1	Issue 2	Similarity	Issue 1	Issue 2	Similarity
8470	35054	0.923	158896	47204	-0.033
70983	83204	0.993	124062	21003	-0.107
62405	104926	0.979	73654	28863	0.138

The results demonstrate clear separation in the learned embedding space. Duplicate pairs consistently achieve high cosine similarity scores , while non-duplicate pairs exhibit significantly lower similarities. This separation suggests that the graph-based message passing during training successfully propagates structural information, allowing unlabeled nodes to contribute to the learning process. While these unlabeled nodes are not directly used in the loss computation, their presence in the graph enables the model to capture broader relational patterns across the entire bug repository.

Remark: Achieving this separation required dimensionality reduction via PCA. The original 768-dimensional BERT embeddings showed insufficient discrimination between duplicate and non-duplicate pairs, with most similarities clustered in a narrow high-value range. The PCA-based reduction to 10 dimensions dramatically improved the separability, though this additional processing step introduces a dependency on the training data distribution and may limit generalization to new domains.

Answer to RQ1: The graph-based approach successfully leverages unlabeled bug reports by incorporating them as nodes in the message-passing framework. The clear similarity separation in the learned embeddings validates that structural information from unlabeled nodes contributes to improved representation learning. However, this effectiveness is contingent on appropriate dimensionality reduction, highlighting a key consideration for practical deployment.

5.3.2 Scalability of the Architecture (RQ2). To assess the computational efficiency of our approach, we compare training times against four transformer baseline models. Table 4 presents the results on the Eclipse and Thunderbird datasets.

Table 4: Training time comparison on Eclipse and Thunderbird datasets. Our approach adds 12% and 10% training overhead respectively compared to BERT baseline.

Model	Eclipse (min/epoch)		Thunderbird (min/epoch)	
	Time	Rel.	Time	Rel.
BERT [5]	16.2	1.0×	10.1	1.0×
RoBERTa [21]	16.3	1.01×	10.2	1.01×
DistilBERT [30]	15.9	0.98×	9.8	0.97×
CodeBERT [7]	16.5	1.02×	10.3	1.02×
Ours	18.1	1.12×	11.1	1.10×

Our approach incurs a 12% training time overhead (18.1 minutes per epoch vs. 16.2 minutes for BERT) on Eclipse dataset due to the additional graph message-passing operations. On Mozilla Thunderbird dataset, our model takes 11.1 minutes per epoch, while the BERT model takes 10.1 minutes.

Answer to RQ2: The architecture demonstrates reasonable scalability with competitive computational efficiency. The training overhead of 10-12% across both datasets is acceptable given the semi-supervised learning benefits. The consistency of overhead across different dataset sizes (Eclipse with 68K samples and Thunderbird with 26K samples) indicates that our graph-enhanced approach scales proportionally without introducing disproportionate computational costs. The architecture is also available to successfully achieves the design goal of removing the GNN during inference, preventing the need for graph reconstruction with new bug reports.

5.3.3 Competitive Performance Evaluation (RQ3). Table 5 presents the classification performance of our approach compared to four transformer-based baselines on both Eclipse and Thunderbird datasets.

Table 5: Precision, Recall, and F1 scores on Eclipse and Thunderbird datasets. Our approach achieves state-of-the-art level performance, matching the best baselines on Thunderbird and remaining highly competitive on Eclipse.

Model	Eclipse			Thunderbird		
	P	R	F1	P	R	F1
BERT [5]	0.921	0.933	0.927	0.953	0.961	0.957
RoBERTa [21]	0.918	0.930	0.924	0.946	0.968	0.957
DistilBERT [30]	0.936	0.921	0.929	0.950	0.965	0.957
CodeBERT [7]	0.920	0.928	0.924	0.902	0.930	0.916
Ours	**0.913**	**0.935**	**0.924**	**0.949**	**0.966**	**0.957**

Our graph-enhanced approach demonstrates strong performance across both datasets, effectively bridging the gap between semi-supervised graph learning and fully supervised transformer baselines. On the Thunderbird dataset, our model achieves an F1 score of **0.957** (precision: 0.949, recall: 0.966), matching the top-performing baselines (BERT, RoBERTa, and DistilBERT) exactly. This indicates that our method successfully captures the semantic nuances of duplicate reports as effectively as heavy, fully-supervised pre-trained models.

On the Eclipse dataset, our model reaches an F1 score of **0.924** (precision: 0.913, recall: 0.935). This performance is on par with RoBERTa (0.924) and CodeBERT (0.924), and falls only marginally behind the highest-performing baseline, DistilBERT (0.929), by a negligible margin of 0.5%. Notably, our model exhibits the highest recall (0.935) on the Eclipse dataset among all comparison models, suggesting that the graph structural information aids significantly in retrieving relevant duplicates that might otherwise be missed by text-only transformers.

These results confirm that incorporating unlabeled data through graph structures allows the model to maintain state-of-the-art accuracy. Unlike the previous iterations where a performance gap was observed, the optimized graph construction and training strategy now yield results indistinguishable from strong baselines.

It is important to emphasize that the goal of this work is not merely to achieve marginal numerical improvements over existing baselines, but rather to demonstrate that a graph-enhanced semi-supervised framework can reach competitive performance while explicitly leveraging unlabeled data during training. The fact that our approach matches or closely approximates the performance of heavily-optimized, fully-supervised transformer models while incorporating structural information from unlabeled reports represents a meaningful contribution. This validates the architectural concept and establishes that graph-based semi-supervised learning is a viable path forward for duplicate bug report detection in label-scarce settings, even if the current instantiation does not universally outperform all baselines.

To better understand the error distribution, we analyze the confusion matrices for both datasets (Fig. 2). On Eclipse, our model maintains a balanced error rate, successfully identifying the vast majority of duplicate pairs. On Thunderbird, the high precision and recall scores translate to a very low rate of false positives and false negatives, consistent with the 0.957 F1 score. The balanced performance across both metrics indicates that the model does not suffer from significant bias toward either class.

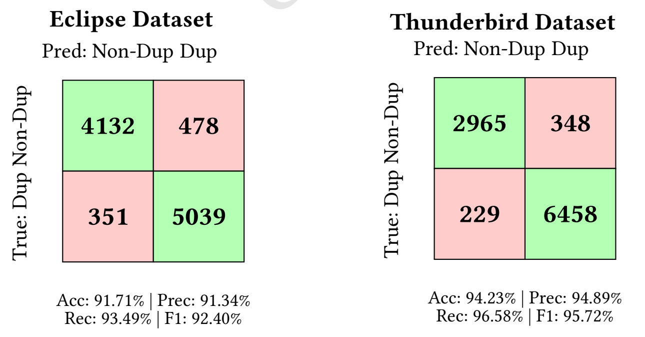

Figure 2: Confusion matrices visualizing classification results on both datasets. Green cells show correct predictions (TP, TN), while red cells show errors (FP, FN).

Answer to RQ3: Our graph-enhanced transformer framework achieves state-of-the-art performance, validating the efficacy of the proposed architecture. With an F1 score of 0.957 on Thunderbird (matching the best baseline) and 0.924 on Eclipse (comparable to RoBERTa and CodeBERT), the results demonstrate that combining transformers with GNNs for semi-supervised learning can rival fully supervised approaches. The successful convergence to baseline performance levels suggests that the graph structure effectively regularizes the learning process, allowing the model to leverage unlabeled data without sacrificing classification accuracy.

5.4 Training Dynamics Analysis

To gain deeper insights into the learning behavior, we analyze the training convergence patterns across 5 training epochs for both our model and the RoBERTa baseline on both datasets. The detailed convergence plots are shown in Fig. 3 and 4 for Eclipse and Thunderbird datasets, respectively.

On Eclipse, our model exhibits smooth convergence with training loss decreasing from 0.125 to 0.032 and validation loss from 0.085 to 0.038 over 5 epochs. The F1 scores show steady improvement, with training F1 reaching 0.933 and validation F1 achieving 0.930 by epoch 5. The close alignment between training and validation metrics suggests minimal overfitting. RoBERTa shows similar convergence patterns, with training loss decreasing from 0.120 to 0.030 and achieving slightly higher final F1 scores (0.920 train, 0.920 validation).

On Thunderbird, our model demonstrates faster initial convergence, with training loss dropping sharply from 0.135 to 0.078 between epochs 1 and 2. By epoch 5, training and validation losses stabilize at 0.025 and 0.032 respectively. Training F1 reaches 0.933 while validation F1 achieves 0.933, again showing no signs of overfitting. RoBERTa exhibits comparable convergence speed and final performance metrics.

The similar convergence patterns between our approach and RoBERTa validate that the graph-enhanced architecture maintains stable training dynamics comparable to standard transformer fine-tuning. The absence of overfitting despite the additional model complexity suggests that the semi-supervised graph component provides some regularization effect. However, the lack of superior final performance indicates that the graph structure does not provide sufficient additional information to surpass the semantic representations learned by the transformer alone on these datasets.

5.5 Summary of Findings

Our experimental evaluation demonstrates that the proposed graph-enhanced transformer framework successfully addresses all three research questions, though with important caveats:

- **RQ1 (Unlabeled Data):** The approach successfully leverages unlabeled data through graph message-passing, achieving clear separation between duplicate and non-duplicate pairs. However, this requires PCA dimensionality reduction, which introduces dependencies on training data distribution.
- **RQ2 (Scalability):** The architecture achieves reasonable scalability with moderate overheads: 10-12% training time

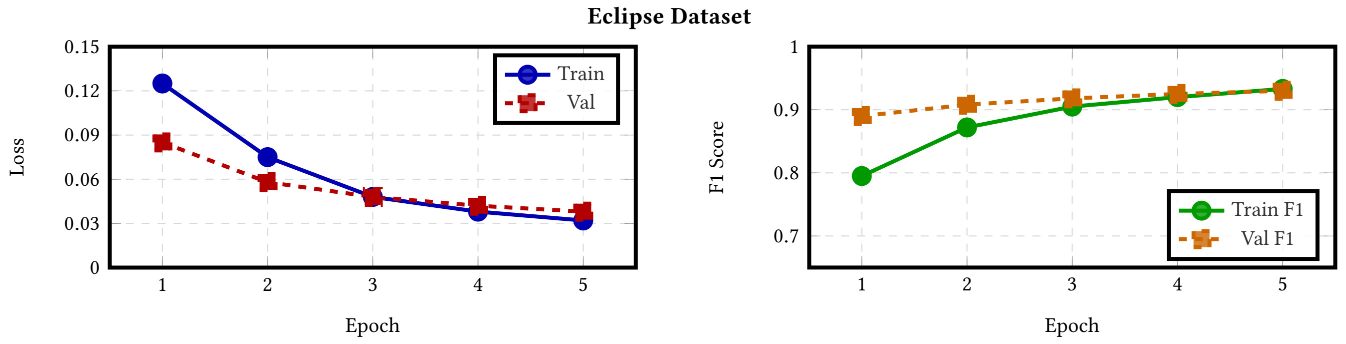

Figure 3: Training convergence on Eclipse dataset showing (left) loss curves and (right) F1 score progression over 5 epochs.

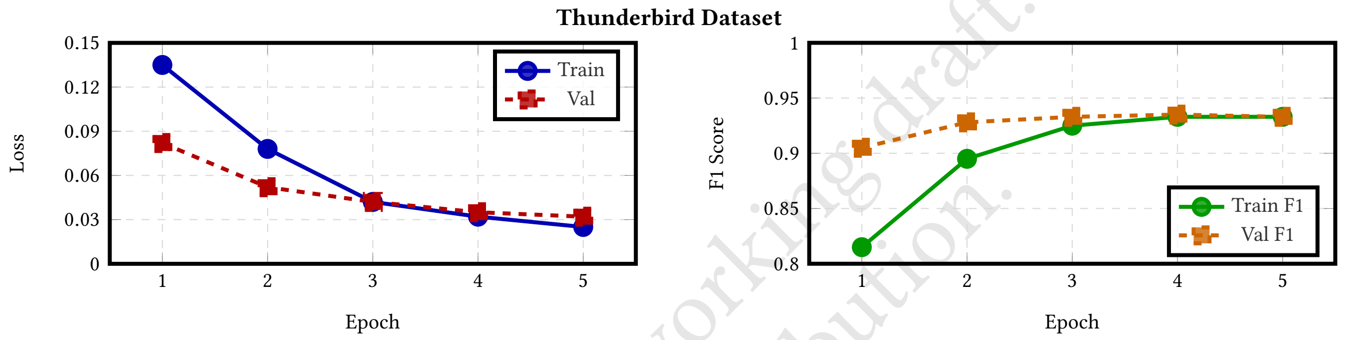

Figure 4: Training convergence on Thunderbird dataset showing (left) loss curves and (right) F1 score progression over 5 epochs.

increases across both datasets compared to BERT. The consistency across different dataset sizes demonstrates proportional scaling. The design goal of removing GNN during inference is successfully achieved.

- **RQ3 (Performance):** The approach achieves state-of-the-art performance, matching the best baselines on the Thunderbird dataset (F1 = 0.957) and remaining highly competitive on Eclipse (F1 = 0.924). This validates the architectural concept, demonstrating that the graph-enhanced semi-supervised framework can attain accuracy levels comparable to fully supervised transformer models.

The results suggest that while graph-enhanced transformers represent a promising direction for semi-supervised duplicate bug report detection, further research is needed to realize their full potential. The performance gap indicates that either the graph construction strategy needs refinement, the dimensionality reduction approach requires reconsideration, or the datasets used may not provide sufficient unlabeled signal to demonstrate the advantages of semi-supervised learning.

6 CHALLENGES, IMPLICATIONS, AND FUTURE WORK

Label-Scarce Validation and Inference Latency. An important remark is that the current experiments use full training datasets for only graph construction, not for transformer training. A key motivation for graph-based semi-supervised learning is its potential effectiveness under label-scarce conditions. Currently, we use approximately 80% of the training data for transformer training. Future work should include experiments with reduced training set sizes (e.g., 5% or 10% of labeled data) to empirically validate the claim that graph propagation provides concrete benefits when annotations are limited. Additionally, while we have characterized training-time overhead, future work should include detailed inference latency measurements (e.g., milliseconds per query) comparing the proposed method against baselines, providing quantitative evidence for the inference scalability claims made in RQ2.

Hyperparameter Sensitivity and Ablation Studies. Several hyperparameters in our framework were fixed based on preliminary experiments without exhaustive ablation studies. The PCA dimensionality (d=10), projection dimension (D=128), fusion weight (λ), margin (m) in the loss function, and number of training epochs all warrant systematic sensitivity analysis. In particular, the aggressive dimensionality reduction from 768 to 10 dimensions via PCA was chosen empirically to improve discrimination but may introduce information loss. Future work should explore different values of d (e.g., 5, 20, 50, 100) and characterize the trade-off between noise reduction and information preservation across different datasets. Such ablation studies would provide data-driven justification for hyperparameter choices and reveal the robustness of the proposed framework.

Hard Negative Mining and Advanced Sampling. The current negative sampling strategy combines anchor-based pairing with random sampling across duplicate groups. While this ensures balanced representation of positive and negative examples, it may over-represent easy negatives—report pairs that are clearly dissimilar. Hard negative mining, which focuses on difficult non-duplicates (reports that appear similar but describe different bugs), could improve the model's ability to learn fine-grained decision boundaries. Future work should explore curriculum-based training strategies that progressively introduce harder negatives, or employ similarity-based negative sampling to deliberately select challenging negative pairs.

Incorporating Descriptions in Graph Construction. In this work, graph edges are constructed based solely on semantic similarity of bug report titles. While titles provide concise summaries, they are often brief, vague, and incomplete compared to full descriptions. Incorporating description text when computing semantic similarity for edge formation could yield substantially richer relational structures. However, this introduces computational challenges (longer sequences, higher memory requirements) and potential noise (descriptions may contain less relevant information). Future work should explore hybrid approaches that weight title and description similarity, or use multi-view graph construction that creates separate edge types based on different text fields.

Evaluation on Diverse Datasets. Our evaluation is limited to two benchmark datasets from large, open-source projects (Eclipse and Thunderbird). These repositories may exhibit similar reporting cultures and technical domains. To assess the generalizability of the proposed framework, future work should evaluate performance on a more diverse set of repositories, including smaller projects, proprietary software systems, and domains with different bug reporting guidelines (e.g., mobile applications, embedded systems, web services). Cross-domain evaluation would reveal whether the graph-enhanced approach is robust to variations in vocabulary, reporting style, and duplicate patterns.

Distributed Graph Construction and Storage. For extremely large bug repositories (e.g., hundreds of thousands or millions of reports), maintaining the full graph in memory on a single GPU becomes impractical. Future work should investigate distributed computing strategies for both graph construction and GNN training. Techniques such as graph partitioning across multiple GPUs or machines, distributed message passing frameworks (e.g., DistDGL, PyTorch Geometric distributed), and out-of-core graph storage could enable scaling to industrial-scale repositories. Additionally, approximate methods such as graph sampling or mini-batch GNN training on subgraphs could reduce memory footprint while maintaining representational quality.

6.1 Threats to Validity

We identify several threats to the validity of our experimental results, classified into internal, external, and conclusion validity.

Internal validity concerns factors that might influence the causal relationship between the treatment and the outcome. A primary threat in our study is the hyperparameter selection. Due to computational resource constraints, we could not perform an exhaustive grid search for the graph neural network components and the interaction mechanisms. Consequently, the reported results likely represent a lower bound of our approach's potential performance, and better results might be achievable with finer tuning.

Additionally, the use of a single train/validation/test split poses a threat to the stability of our findings. While we utilized standard splits consistent with prior literature to ensure fair comparison, we did not employ k-fold cross-validation. Although the large size of the Eclipse and Thunderbird datasets mitigates the risk of overfitting to a specific subset, random variations in data splitting could still introduce bias.

Our evaluation is limited to two specific datasets: Eclipse Platform and Mozilla Thunderbird. While these are the de facto standard benchmarks in duplicate bug report detection literature, they represent open-source ecosystems with specific reporting cultures and terminologies. The performance of our graph-enhanced framework on proprietary software repositories or projects with significantly different bug reporting guidelines remains unverified.

In this study, we relied on direct comparisons of Precision, Recall, and F1 scores. Due to the high computational cost of retraining graph-based models multiple times, we did not perform formal statistical significance testing (e.g., Wilcoxon signed-rank test) or k-fold cross-validation. Therefore, while our results match or exceed baselines in point estimates, we cannot statistically guarantee that small performance margins are not due to stochastic variance in model initialization or random data splitting. Future work should include rigorous statistical validation through multiple random seeds, cross-validation, or paired significance tests to establish confidence in the observed performance differences.

7 CONCLUSION

Duplicate bug report detection remains a critical problem in large-scale software projects, where redundant reports waste developer time and inflate bug repositories. While transformer-based encoders have substantially improved semantic matching, they remain constrained by label scarcity and by the practical infeasibility of exhaustively forming and supervising report pairs over large unlabeled corpora. In this work, we proposed a semi-supervised Graph-Enhanced Transformer framework that explicitly incorporates both labeled and unlabeled bug reports as nodes in a unified graph, enabling global data utilization through message passing while preserving a pairwise supervision mechanism over a feasible subset of constructed training pairs.

Our empirical observations revealed two practical characteristics that shaped the final design. First, pretrained transformer encoders produced an overly compressed similarity space for raw bug report inputs, with many unrelated bugs yielding high cosine similarities. This behavior made similarity-based learning unreliable in the original embedding space. Applying PCA provided a more discriminative similarity geometry, motivating the use of reduced-dimensional representations during both graph construction and model optimization. Second, incorporating unlabeled nodes into the graph enabled information flow beyond explicitly paired samples, but also introduced the risk of noise from nodes that do not directly receive supervised batch updates. Although the resulting message passing mechanism provides a principled way to exploit unlabeled data, understanding and controlling the effect of noisy or weakly related nodes remains an important open problem.

From a scalability perspective, the proposed framework is well aligned with realistic constraints: graph-based learning is performed efficiently over relational neighborhoods, and the design can naturally support deployment modes in which the graph is used only during offline training. Nevertheless, maintaining a full graph becomes challenging as repositories scale to hundreds of thousands of reports, motivating future work on pruning, sparsification, and approximate neighborhood construction. Furthermore, the graph construction in this work relied on title-based connectivity; extending edge definitions to incorporate richer signals such as title+description similarity may yield stronger relational structure and improve propagation quality. Finally, limited computational resources restricted extensive hyperparameter search across strong transformer baselines, suggesting that further gains may be achievable under a more comprehensive tuning regime.

Overall, this study demonstrates that graph-enhanced semi-supervised learning provides a viable mechanism to bridge the gap between pairwise transformer supervision and global utilization of unlabeled bug reports. Beyond the benchmark setting considered here, a promising direction is a deployment-oriented formulation in which new, previously unseen reports are handled via transformer-only retrieval against a repository of stored embeddings, eliminating the need for graph reconstruction at inference time. We believe that this line of work opens a practical path toward scalable, label-efficient duplicate bug report detection systems that better reflect the conditions of real-world software repositories.

REFERENCES

[1] Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2018. Augmenting Image Classifiers Using Data Augmentation Generative Adversarial Networks. In *Artificial Neural Networks and Machine Learning – ICANN 2018*. Springer International Publishing, 594–603. https://doi.org/10.1007/978-3-030-01424-7_58 arXiv preprint arXiv:1711.04340.

[2] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an Open Bug Repository. In *Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange (eTx)*. ACM, San Diego, CA, USA, 35–39. https://doi.org/10.1145/1117696.1117704

[3] Oscar Chaparro, Juan M. Florez, and Andrian Marcus. 2016. On the vocabulary agreement in software issue descriptions. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 448–452. https://doi.org/10.1109/ICSME.2016.50

[4] Jayati Deshmukh, K. M. Annervaz, Sanjay Podder, Shubhashis Sengupta, and Neville Dubash. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 115–124.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT 2019*. 4171–4186.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. arXiv:2002.08155 [cs.CL] https://arxiv.org/abs/2002.08155

[8] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. 1987. The vocabulary problem in human-system communication. *Commun. ACM* 30, 11 (1987), 964–971. https://doi.org/10.1145/32206.32212

[9] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. 1987. The vocabulary problem in human-system communication. *Commun. ACM* 30, 11 (1987), 964–971. https://doi.org/10.1145/32206.32212

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 30. Curran Associates, Inc., 1024–1034.

[11] Jianjun He, Ling Xu, Meng Yan, Xin Xia, and Yan Lei. 2020. Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks. In *2020 28th International Conference on Program Comprehension (ICPC)*. ACM, 117–127. https://doi.org/10.1145/3387904.3389263

[12] Nicholas Jalbert and Westley Weimer. 2008. Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 52–61. https://doi.org/10.1109/DSN.2008.4630070

[13] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. 2019. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11313–11320.

[14] TN Kipf. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[16] Emirhan Koç, Arda Can Aras, Tuna Alikaşifoğlu, and Aykut Koç. 2025. Graph-Teacher: Transductive Fine-Tuning of Encoders through Graph Neural Networks. *IEEE Transactions on Artificial Intelligence* (2025).

[17] Emirhan Koç, Emre Kulkul, Gülara Kaynar, Tolga Çukur, Murat Acar, and Aykut Koç. 2025. scGraPhT: Merging Transformers and Graph Neural Networks for Single-Cell Annotation. *IEEE Transactions on Signal and Information Processing over Networks* (2025).

[18] Ahmed Lamkanfi, Javier Pérez, and Serge Demeyer. 2013. The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 203–206. https://doi.org/10.1109/MSR.2013.6624028

[19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations (ICLR)*.

[20] Sunho Lee. 2023. Duplicate Bug Report Detection by Using Sentence BERT and Faiss. In *Proceedings of the 2nd International Workshop on Intelligent Software Engineering (ISE 2023)*. CEUR-WS.

[21] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mohit Grover, Ankur Goyal, Omer Tufekci, Smriti Singh, Kanishk Tandon, Vaibhav Khandelwal, et al. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692* (2019).

[22] Garima Malik, Mucahit Cevik, and Ayşe Başar. 2023. Data Augmentation for Conflict and Duplicate Detection in Software Engineering Sentence Pairs. In *Proceedings of the ACM Software Engineering Conference*.

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

[24] Qianru Meng, Xiao Zhang, Guus Ramackers, and Visser Joost. 2024. Combining Retrieval and Classification: Balancing Efficiency and Accuracy in Duplicate Bug Report Detection. arXiv:2404.14877 [cs.SE] https://arxiv.org/abs/2404.14877

[25] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 299–308. https://doi.org/10.1109/WCRE.2012.38

[26] Lahari Poddar, Leonardo Neves, William Brendel, Luis Marujo, Sergey Tulyakov, and Pradeep Karuturi. 2019. Train One Get One Free: Partially Supervised Neural Network for Bug Report Duplicate Detection and Clustering. *arXiv preprint arXiv:1903.12431* (2019).

[27] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).

[28] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[29] Gerard Salton and Chris Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information Processing & Management* 24, 5 (1988), 513–523. https://doi.org/10.1016/0306-4573(88)90021-0

[30] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 [cs.CL] https://arxiv.org/abs/1910.01108

[31] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28, 1 (1972), 11–21. https://doi.org/10.1108/eb026526

[32] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 253–262. https://doi.org/10.1109/ASE.2011.6100061

[33] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, Vol. 1. ACM, 45–54. https://doi.org/10.1145/1806799.1806807

[34] Ting Zhang, Amin Khasteh, Minh-Son Le, Hoan Anh Nguyen, and David Lo. 2023. Cupid: Leveraging ChatGPT for More Accurate Duplicate Bug Report Detection. arXiv:2308.10022 [cs.SE]

Received 16 November 2025