

# Graph Enhanced Transformer for Semi-Supervised Duplicate Bug Report Detection

1 Kerem Bayramoğlu\*

2 kerem.bayramoglu@bilkent.edu.tr  
3 Bilkent University  
4 Dept. of Computer Engineering  
5 Çankaya, Ankara, Türkiye

6 Hüseyin Karaca†

7 huseyin.karaca@bilkent.edu.tr  
8 Bilkent University  
9 Dept. of Electrical and Electronics  
10 Engineering  
11 Çankaya, Ankara, Türkiye

12 Emirhan Koç†

13 emirhan.koc@bilkent.edu.tr  
14 Bilkent University  
15 Dept. of Electrical and Electronics  
16 Engineering  
17 Çankaya, Ankara, Türkiye

18 Hasan Erkin Ünlü†

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

20 Rabia Ela Ünlü†

21 elua.unlu@bilkent.edu.tr  
Bilkent University  
Dept. of Electrical and Electronics  
Engineering  
Çankaya, Ankara, Türkiye

## ABSTRACT

Context. Duplicate bug report detection aims to identify issue 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 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.

\*Authors are listed in alphabetical order by surname.

Unpublished working draft. Not for distribution.  
Permission to make digital or hard copies of all or part of this work for personal or  
internal use is granted without fee provided that copies are not made or distributed  
for profit or commercial advantage and that copies bear this notice and the full citation  
on the first page. Copyrights for components of this work owned by others than the  
author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or  
repubish, to post on servers or to redistribute to lists, requires prior specific permission  
and/or a fee. Request permissions from permissions@acm.org.

CS 588, Fall 2025-26, Bilkent University

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06

<https://doi.org/XXXXXXX.XXXXXXX>

2026-01-02 11:33. Page 1 of 1-12.

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.<sup>1</sup>

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 training phase and retaining transformer-only similarity inference at test time, 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.

## KEYWORDS

Issue, 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, 12 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 INTRODUCTION

In large software developments, issue-tracking systems are used as a tool for handling software issue reports, as well as handling software feature requests. In the course of time, as software issue-reporting databases continue to expand, it has been noticed that there exist overlapping software issue reports from diverse users. This results in the enlargement of the size of the software issue-reporting databases, as well as consuming the time of software

<sup>1</sup><https://github.com/huseyin-karaca/graph-enhanced-dbd.git>

117 developers in examining the same software issues, as these overlapping software issue reports are nothing but software duplicates.  
 118 Studies show that a large number of bug duplicates exist in bug  
 119 repositories, for example for example, 20% of reports in Eclipse and  
 120 30% in Firefox were marked as duplicate [2].  
 121

122 Automating duplicate bug report detection has therefore become  
 123 an essential research direction in software engineering. Classical in-  
 124 formation retrieval (IR) techniques such as Term Frequency–Inverse  
 125 Document Frequency (TF-IDF), Best Matching 25 with field weight-  
 126 ing (BM25F), and custom retrieval functions (e.g., REP) have been  
 127 widely used to match new bug reports with existing ones [32].  
 128

129 Although these algorithms are computationally efficient, they  
 130 remain limited in addressing the vocabulary mismatch problem,  
 131 where semantically similar bug reports are expressed using different  
 132 lexical choices [8]. More recent approaches based on deep learning,  
 133 particularly transformer-based models derived from BERT, have  
 134 demonstrated improved effectiveness in capturing semantic similar-  
 135 ity by leveraging contextual representations of bug report titles and  
 136 descriptions [5, 20, 27]. Despite these advances, transformer-based  
 137 methods rely heavily on supervised learning with labeled duplicate  
 138 reports, which introduces a substantial dependency on annotated  
 139 data. In practice, acquiring such labels is costly and labor-intensive,  
 140 and real-world bug repositories typically contain only a limited  
 141 number of confirmed duplicate annotations. To mitigate the re-  
 142 liance on limited annotated data, several studies have proposed  
 143 augmenting supervision by exploiting unlabeled bug reports, for in-  
 144 stance through pairing strategies combined with negative sampling  
 145 [22, 26]. While such approaches partially alleviate label scarcity,  
 146 they remain limited in their ability to fully leverage the entire pool  
 147 of available unlabeled data, and thus do not provide a comprehen-  
 148 sive solution for large-scale, label-sparse bug repositories.  
 149

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

165 It is worth emphasizing that this work represents an early effort  
 166 to explore the use of GNNs for duplicate bug report detection,  
 167 while simultaneously revealing several promising directions for  
 168 future improvement. Although the proposed framework adopts a  
 169 transductive training and inference setting by constructing a graph  
 170 that includes all nodes from the training, validation, and test splits,  
 171 its design is not inherently restricted to this regime. With increased  
 172 model capacity and appropriate architectural modifications, the  
 173 framework can be naturally extended to support inductive inference,  
 174 enabling generalization to previously unseen bug reports.

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? 175  
 176
- **RQ2:** Can a scalable architecture be designed to support duplicate bug report detection in large-scale bug repositories? 177  
 178
- **RQ3:** Does the proposed graph-enhanced, semi-supervised framework achieve competitive performance compared to strong transformer-based baselines? 179  
 180

181 This study offers the following contributions:  
 182

- 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. 183  
 184
- 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. 185  
 186
- 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. 187  
 188
- 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. 189  
 190

## 2 PROBLEM DESCRIPTION AND MOTIVATION

Transformer-based sentence encoders achieve strong semantic representations but are inherently limited in their ability to exploit unlabeled data beyond implicit pretraining. In practical duplicate bug report detection settings, this limitation becomes critical, as labeled duplicate pairs are scarce while the majority of available bug reports remain unlabeled. Directly incorporating all unlabeled bug reports into transformer-based pairwise training is computationally infeasible and methodologically ill-defined, as supervision is only available for a small subset of report pairs.

To overcome this limitation, we leverage the structural flexibility of graph-based learning. Specifically, we construct a graph in which all available bug reports, including both labeled and unlabeled instances, are represented as nodes. Edges are formed based on label-independent semantic relationships, enabling efficient construction of the graph without requiring additional annotations. Since graph neural networks operate on relational neighborhoods rather than explicit pairwise supervision, this formulation allows the full bug report collection to be incorporated into training in a scalable manner.

During training, the transformer component operates on a restricted but feasible subset of bug report pairs. Positive pairs are formed from known duplicate bug reports, while negative pairs are generated by sampling from the unlabeled pool, following standard practice in real-world DBRD scenarios where explicit negative labels are unavailable. The representations learned from these labeled and pseudo-labeled pairs are then propagated to their corresponding nodes in the graph. Although direct supervision is applied only

233 to nodes participating in these pairs, the GNN enables information  
 234 to flow beyond them through message passing, allowing unpaired  
 235 and unlabeled nodes to indirectly contribute to representation learning.  
 236

237 As a result, bug reports that are never explicitly paired or as-  
 238 signed labels are not excluded from the learning process. Instead,  
 239 they influence and are influenced by neighboring nodes through  
 240 graph propagation, effectively injecting their semantic and struc-  
 241 tural context into the model. This design decouples pairwise su-  
 242 pervision from global data utilization: while the transformer is  
 243 trained on a manageable subset of labeled and pseudo-labeled pairs,  
 244 the graph component enables the model to benefit from the full  
 245 unlabeled corpus. Consequently, the proposed framework bridges  
 246 the gap between transformer-based semantic modeling and large-  
 247 scale semi-supervised learning, making it more suitable for realistic  
 248 duplicate bug report detection settings.  
 249

### 250 3 RELATED WORK

251 The problem of duplicate bug report detection has been a central  
 252 challenge in software engineering research for many years. Early  
 253 approaches primarily relied on classical Information Retrieval (IR)  
 254 methods, while more recent techniques have leveraged advances  
 255 in machine learning and deep learning to improve performance. In  
 256 this section, we review the evolution of methodologies for DBRD,  
 257 highlighting key contributions and their limitations.  
 258

#### 259 3.1 Information Retrieval Models

260 Early automated approaches framed DBRD as a classical Information  
 261 Retrieval (IR) problem [33], typically using the Vector Space  
 262 Model (VSM) [23]. In this model, each bug report is treated as a  
 263 document and is transformed into a high-dimensional vector. The  
 264 components of this vector are weighted based on the terms present  
 265 in the document.  
 266

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

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

#### 276 3.2 Machine Learning Approaches

277 In response, researchers applied supervised machine learning, shifting  
 278 the problem from retrieval to classification (a binary duplicate/non-  
 279 duplicate decision) [12, 33]. Early work, such as Jalbert and Weimer  
 280 (2008) [12] and Sun et al. (2010) [33], applied classifiers like Support  
 281 Vector Machines (SVMs) to feature pairs.  
 282

283 However, training a classifier requires generating  $O(N^2)$  po-  
 284 tential pairs, which is computationally intractable [33]. This "low  
 285 efficiency" of pair-wise classification led to a critical development:  
 286 using machine learning to improve the IR model itself.

287 This led to REP, a retrieval function proposed by Sun et al. [32]  
 288 that became a dominant baseline. REP extends the Okapi BM25F  
 289 formula to create a custom similarity function. Its key innovation

290 is combining textual similarity with structured metadata (product,  
 291 component, version) [32]. It then uses supervised learning to learn  
 292 the optimal weights for each field, tuning its function for a specific  
 293 bug repository [32]. The success of REP demonstrated that a spe-  
 294 cialized, feature-aware IR model could outperform general-purpose  
 295 ML models in both accuracy and efficiency [32, 33], and it remained  
 296 a difficult baseline to beat for years [32].  
 297

### 298 3.3 Deep Learning for Semantic Representation

299 The deep learning (DL) revolution promised to finally solve the  
 300 vocabulary mismatch problem by learning true semantic meaning  
 301 [3]. Initial attempts with Convolutional Neural Networks (CNNs)  
 302 and Recurrent Neural Networks (RNNs), such as the Dual-Channel  
 303 CNN (DCCNN) by He et al. (2020) [11], began learning vector  
 304 embeddings from report text. However, these early DL models  
 305 often struggled to outperform the highly-optimized REP baseline  
 306 [32].  
 307

308 A significant shift occurred with the Transformer model, specifi-  
 309 cally BERT [6]. BERT's pre-training allows it to capture deep, con-  
 310 textual understanding of language [6]. This base model was quickly  
 311 followed by optimized variants such as RoBERTa [21], which im-  
 312 proved performance by refining the pre-training process, and AL-  
 313 BERT [19], which focused on parameter reduction. However, apply-  
 314 ing these cross-encoder models (BERT, RoBERTa, ALBERT) directly  
 315 to DBRD exposed a critical flaw. As a cross-encoder, BERT requires  
 316 both reports to be fed into the network simultaneously [28]. Finding  
 317 the best match for a new report in a large database would require  
 318 10,000s of computations, a process estimated to take around 65  
 319 hours [28], rendering it unusable.  
 320

321 This was solved by Reimers and Gurevych (2019) with Sentence-  
 322 BERT (SBERT) [28]. SBERT adapts BERT using a siamese network  
 323 structure [28]. Two identical BERT models process each bug report  
 324 independently, producing a single "sentence embedding" for the pair  
 325 [28]. Because embeddings are generated independently, they can be  
 326 pre-computed. A new report can be embedded once, and its simila-  
 327 rity to all others found almost instantaneously via cosine similarity  
 328 search [28]. This reduces the 65-hour search task to approximately  
 329 5 seconds [28]. SBERT provides a powerful, semantic-native re-  
 330 placement for TF-IDF, solidifying the dominance of the two-stage  
 331 "retrieval-rerank" pipeline. A recent comparative study by Meng et  
 332 al. (2024) empirically evaluated many of these architectures, con-  
 333 firming the performance gains of transformer-based models like  
 334 BERT, ALBERT, and RoBERTa over earlier neural architectures like  
 335 Bi-LSTM and DC-CNN [24].  
 336

### 337 3.4 Hybrid Systems and Emerging Approaches

338 Current state-of-the-art systems are often hybrid approaches com-  
 339 bining lexical precision with semantic understanding [25]. The  
 340 DBTM (Duplicate Bug report Topic Model) approach, for example,  
 341 combines the IR model BM25F with topic-based features [25], al-  
 342 lowing it to match reports based on higher-level "technical issues"  
 343 [25].  
 344

345 More recently, this philosophy has extended to Large Language  
 346 Models (LLMs). Cupid [34] achieves state-of-the-art results by com-  
 347 bining an LLM (ChatGPT) with the classic IR model REP [32, 34].  
 348 The LLM is used in a zero-shot setting as a semantic pre-processor  
 349

349 to "get essential information" from the raw bug report [34]. This  
 350 "cleaned" information is then fed to the domain-aware REP model  
 351 for the final similarity calculation. This hybrid approach was shown  
 352 to improve recall over previous baselines by a significant margin  
 353 [34].

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

364 Our proposed framework addresses these two specific, critical  
 365 gaps: (1) label scarcity and (2) inference scalability. We operate in a  
 366 *semi-supervised* setting, using a GNN to leverage the vast majority  
 367 of *unlabeled* reports during training. Crucially, our method remains  
 368 *inductive* and scalable by design: the GNN is used only as a training-  
 369 time enhancement and is discarded for inference, where only the  
 370 efficient transformer encoder is needed. This hybrid approach aims  
 371 to achieve the semantic richness of deep transformers, enhanced  
 372 by the relational context from unlabeled data, while preserving the  
 373 high-speed inference of an SBERT-like bi-encoder. Table 1 outlines  
 374 this positioning relative to closely related works.

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

## 385 4 PROPOSED METHOD

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

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

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

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

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

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

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

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

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

420 where  $\alpha$  denotes the fixed subsampling ratio.

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

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

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

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

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

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

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

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

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

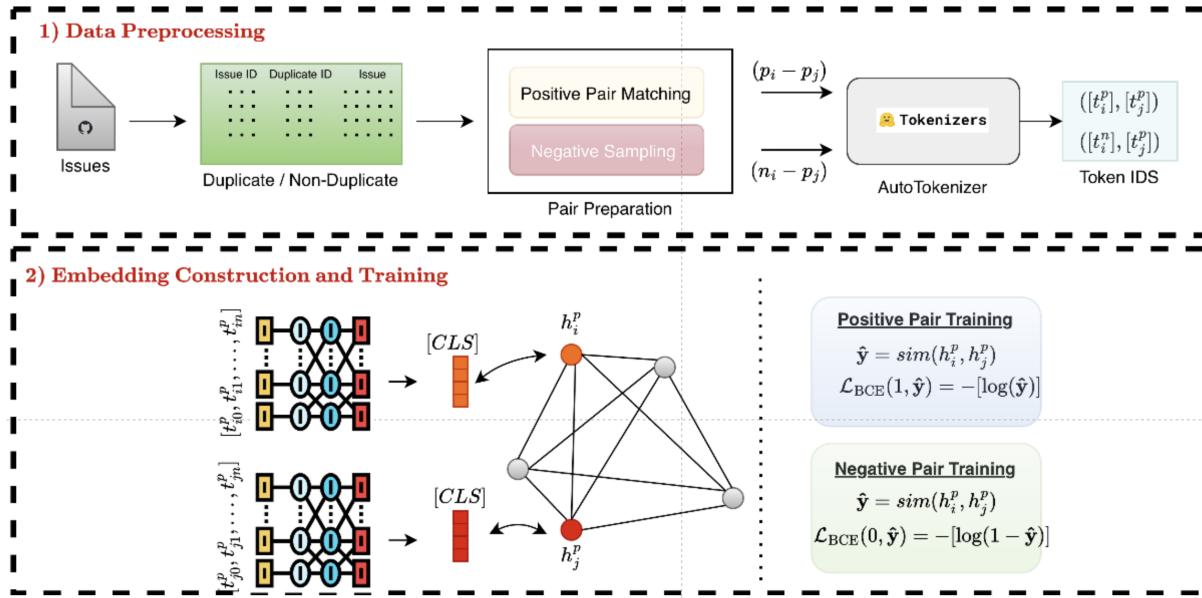
### 4.1 Embedding and Graph Construction

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

455 *4.1.1 Graph Construction.* GNNs provide a flexible representation  
 456 learning framework capable of operating on both labeled and un-  
 457 labeled nodes. A GNN defines a message-passing operator over  
 458 a graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where information is iteratively exchanged  
 459 among neighboring nodes to produce context-aware node embed-  
 460 dings [13, 14]. Since the update functions in a GNN do not rely on  
 461 node labels, the model naturally supports semi-supervised settings  
 462 in which only a subset of nodes is labeled while the remaining nodes  
 463 remain unlabeled [16]. This property is particularly well aligned

**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)
<b>Our Method</b>	<b>Transformer + GNN (Hybrid)</b>	<b>Deep Contextual &amp; Relational</b>	<b>Yes (Semi-supervised)</b>	<b>High (Bi-encoder)</b>



**Figure 1: Overall pipeline of the proposed method.** Upper: The issues are documented with issue numbers, titles, and descriptions. The positive and negative pairs are collected and stacked. Then, the token id's are listed to be fed into BERT. Middle: 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.

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 issue 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 issues; 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 issue 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 issue titles. Let  $\tau_i$  denote the tokenized title of issue  $i$ . 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.

**581** *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.

**590** Formally, given a token-ID sequence

$$\mathbf{591} \quad t^{(i)} = [t_0^{(i)}, \dots, t_{L_i}^{(i)}],$$

**593** the pretrained transformer encoder is denoted by

$$\mathbf{595} \quad f_W : \mathbb{N}^{L_i} \rightarrow \mathbb{R}^{L_i \times d},$$

**597** 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

$$\mathbf{600} \quad H_i = f_W(t^{(i)}) \in \mathbb{R}^{L_i \times d}.$$

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

$$\mathbf{604} \quad h_i^{(\text{CLS})} = H_i[0] \in \mathbb{R}^d,$$

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

**608** 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:

$$\mathbf{612} \quad z_i = \mathbf{W}_p h_i^{(\text{CLS})} + \mathbf{b}_p, \quad \mathbf{W}_p \in \mathbb{R}^{D \times d},$$

**614** 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.

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

$$\mathbf{620} \quad z_i = \mathbf{W}_p f_W(t^{(i)})[0], \quad z_j = \mathbf{W}_p f_W(t^{(j)})[0].$$

**621** 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.

**626** *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.

**639** *Fusion of Transformer and GNN representations.* Let  $z_i^{(\text{tr})} \in \mathbb{R}^D$  denote the projected transformer embedding of report  $i$ , and let **640**  $z_i^{(\text{gnn})} \in \mathbb{R}^D$  denote the corresponding GNN output embedding **641** after message passing. The final embedding used for optimization **642** is obtained via a weighted fusion:

$$\mathbf{644} \quad z_i = \lambda z_i^{(\text{tr})} + (1 - \lambda) z_i^{(\text{gnn})}, \quad \lambda \in [0, 1].$$

**647** This formulation balances semantic information captured by the **648** transformer with relational information captured by the GNN. As **649** an alternative design choice, the two embeddings may also be combined **650** via concatenation, i.e.,  $z_i = [z_i^{(\text{tr})}; z_i^{(\text{gnn})}]$ , followed by a **651** projection layer. Unless otherwise stated, the weighted-sum fusion **652** is used throughout this work.

**653** *Cosine embedding objective.* For each supervised report pair  $(i, j)$ , **654** we compute the cosine similarity

$$\mathbf{656} \quad s_{ij} = \cos(z_i, z_j).$$

**658** We optimize a cosine embedding loss, where pairwise labels are **659** mapped to  $\{-1, +1\}$ . Specifically, positive (duplicate) pairs are **660** assigned  $y_{ij} = +1$ , while sampled negative pairs are assigned  $y_{ij} = -1$ . **661** The loss for a pair  $(i, j)$  is defined as

$$\mathbf{663} \quad \mathcal{L}_{ij} = \begin{cases} 1 - s_{ij}, & y_{ij} = +1, \\ \max(0, s_{ij} - m), & y_{ij} = -1, \end{cases}$$

**666** where  $m \in [0, 1]$  denotes a margin hyperparameter. The overall **667** training objective is given by

$$\mathbf{669} \quad \mathcal{L} = \sum_{(i,j) \in \mathcal{P}^{(+)}} \mathcal{L}_{ij} + \sum_{(i,j) \in \mathcal{N}} \mathcal{L}_{ij}.$$

**672** Minimizing  $\mathcal{L}$  jointly updates the parameters of the transformer **673** encoder, the projection module, and the GNN, enabling similarity-**674** preserving representations informed by both textual semantics and **675** graph-structured relational information.

**676** *Validation-based decision rule.* During validation, similarity scores **677** are computed only for pairs involving nodes belonging to the **678** validation split. A pair  $(i, j)$  is predicted as duplicate if its similarity **679** score exceeds a threshold  $\theta$ :

$$\mathbf{681} \quad \hat{y}_{ij} = \mathbb{I}[s_{ij} \geq \theta].$$

**684** A default threshold of  $\theta = 0.5$  is used as an initial operating point **685** during training. To determine an appropriate decision threshold, **686** we perform a validation-time grid search over  $\theta \in [0, 1]$  with a **687** step size of 0.05, and select the value that yields the best validation **688** performance. The selected threshold is then fixed for subsequent **689** evaluation.

## 4.2 Inference Procedure

**691** Let  $\hat{f}_W$  and  $\hat{g}_\Theta$  denote the transformer encoder (including projection) **692** and the GNN with parameters fixed after training. During **693** inference, all model parameters are frozen and no further updates **694** 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

$$z_i = \hat{f}_W(t^{(i)}), \quad z_j = \hat{f}_W(t^{(j)}),$$

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  $\theta$ , yielding the prediction

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

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.

**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

### 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

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

Using these quantities, accuracy is computed as

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

Precision and recall are defined as

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

and the F1-score is given by

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

**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  $2 \times 10^{-5}$  and a batch size of 64. Models are trained for up to 5 epochs with early stopping based on validation performance. 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 issue 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 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 the codes and full reproducibility package 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	<b>0.923</b>	158896	47204	-0.033
70983	83204	<b>0.993</b>	124062	21003	-0.107
62405	104926	<b>0.979</b>	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×
<b>Ours</b>	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 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
<b>Ours</b>	<b>0.913</b>	<b>0.935</b>	<b>0.924</b>	<b>0.949</b>	<b>0.966</b>	<b>0.957</b>

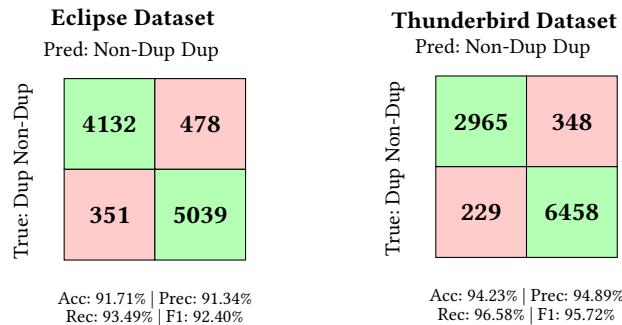
Our graph-enhanced approach demonstrates strong performance across both datasets, effectively bridging the gap between semi-supervised graph learning and fully supervised transformer baselines. On the Thunderbird dataset, our model achieves an F1 score of **0.957** (precision: 0.949, recall: 0.966), 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.

To better understand the error distribution, we analyze the confusion matrices for both datasets (Figure 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.

*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



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

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.

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 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

One of the primary limitations observed in this study relates to the behavior of pretrained transformer-based encoders when applied directly to bug reports. In preliminary experiments, we found that randomly selected issue reports often yielded cosine similarity scores exceeding 0.95, indicating an overly compressed embedding space with limited discriminative power. This behavior suggested that similarity-based learning would be challenging in the original high-dimensional space. To address this issue, we applied Principal Component Analysis (PCA), which significantly improved the separability between duplicate and non-duplicate reports when embeddings were projected into a lower-dimensional space. Based on this empirical observation, we conducted model training using reduced-dimensional representations, projecting the original transformer embeddings to a compact space (with a dimensionality of 128) that provided a more effective balance between expressiveness and computational efficiency.

A second limitation concerns the graph construction process, particularly the inclusion of unlabeled nodes that do not directly participate in supervised batch updates. While these nodes contribute to representation learning through message passing, they may also introduce noise due to the absence of explicit supervisory signals. Although the GNN framework enables such unlabeled nodes to influence the learning process indirectly, the extent to which noisy or weakly related nodes affect overall performance remains insufficiently understood and warrants further investigation.

Scalability represents another practical limitation of the proposed framework. In scenarios where the number of bug reports

929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044

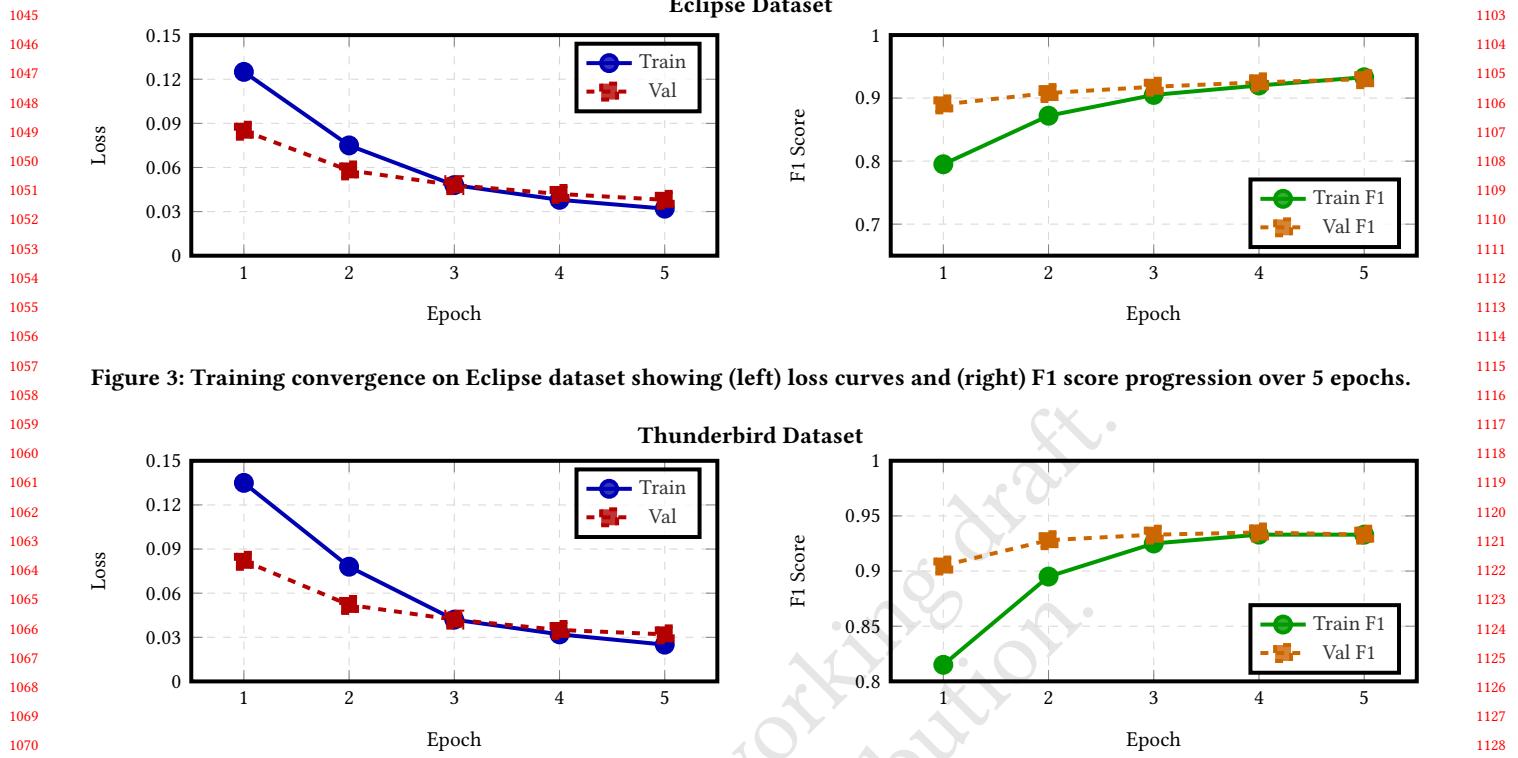


Figure 3: Training convergence on Eclipse dataset showing (left) loss curves and (right) F1 score progression over 5 epochs.

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

grows substantially beyond the scale considered in this work, maintaining the full graph in memory may become infeasible. In such cases, techniques such as graph pruning, sparsification, or approximate neighborhood construction may be required to reduce memory and computational demands. Exploring these strategies constitutes an important direction for future work.

Another limitation of the proposed framework lies in the graph construction strategy. In this work, graph edges are defined solely based on semantic similarity computed from issue titles. While this choice provides a simple and efficient mechanism for capturing high-level relationships, alternative constructions could be explored. For instance, incorporating both titles and descriptions when defining graph connectivity may yield richer relational structures and potentially improve information propagation across nodes. Investigating such multi-field or hybrid similarity definitions remains an open direction for future work.

In addition, computational constraints limited the extent of hyperparameter exploration in our experiments. Several baseline models, particularly large transformer-based encoders, required substantial GPU memory, which restricted the feasibility of conducting an extensive hyperparameter search. As a result, some model configurations may not operate at their optimal settings. This limitation likely affects the overall performance and suggests that further gains could be achieved with more comprehensive tuning under less restrictive computational resources.

The current framework assumes a graph constructed over a fixed set of reports. In real-world deployment scenarios, newly

submitted bug reports may arrive without existing connections to the graph. Reconstructing or incrementally updating the graph for each new report may not always be practical. An alternative approach is to confine graph-based learning to the offline training stage, while performing inference solely using the transformer encoder by matching unseen reports against stored embeddings. Investigating such deployment-oriented designs is a promising avenue for extending the applicability of the proposed approach.

## 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). 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.

## 7 CONCLUSION

Duplicate bug report detection remains a critical problem in large-scale software projects, where redundant reports waste developer time and inflate issue 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 issues 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](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, Ziyian 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.

- 1277      *IEEE Transactions on Artificial Intelligence* (2025).
- 1278 [17] Emirhan Koç, Emre Kulkul, Güleray Kaynar, Tolga Çukur, Murat Acar, and Aykut  
1279      Koç. 2025. scGraPhT: Merging Transformers and Graph Neural Networks for  
1280      Single-Cell Annotation. *IEEE Transactions on Signal and Information Processing  
over Networks* (2025).
- 1281 [18] Ahmed Lamkanfi, Javier Pérez, and Serge Demeyer. 2013. The Eclipse and  
1282      Mozilla defect tracking dataset: A genuine dataset for mining bug information.  
1283      In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 203–206.  
1284      <https://doi.org/10.1109/MSR.2013.6624028>
- 1285 [19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush  
1286      Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised  
1287      Learning of Language Representations. In *International Conference on Learning  
1288      Representations (ICLR)*.
- 1289 [20] Sunho Lee. 2023. Duplicate Bug Report Detection by Using Sentence BERT and  
1290      Faiss. In *Proceedings of the 2nd International Workshop on Intelligent Software  
1291      Engineering (ISE 2023)*. CEUR-WS.
- 1292 [21] Yinhai Liu, Myla Ott, Namara Goyal, Jingfei Du, Mohit Grover, Ankur Goyal,  
1293      Omer Tufekci, Smriti Singh, Kanishk Tandon, Vaibhav Khandelwal, et al. 2019.  
1294      RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint  
1295      arXiv:1907.11692* (2019).
- 1296 [22] Garima Malik, Mucahit Cevik, and Ayşe Başar. 2023. Data Augmentation for  
1297      Conflict and Duplicate Detection in Software Engineering Sentence Pairs. In  
1298      *Proceedings of the ACM Software Engineering Conference*.
- 1299 [23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge,  
1300      UK.
- 1301 [24] Qianru Meng, Xiao Zhang, Guus Ramackers, and Visser Joost. 2024. Combining  
1302      Retrieval and Classification: Balancing Efficiency and Accuracy in Duplicate Bug  
1303      Report Detection. *arXiv:2404.14877 [cs.SE]* <https://arxiv.org/abs/2404.14877>
- 1304 [25] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, and Cheng-  
1305      nian Sun. 2012. Duplicate bug report detection with a combination of information  
1306      retrieval and topic modeling. In *Proceedings of the 20th Working Conference on Re-  
1307      verse Engineering (WCRE)*. IEEE, 299–308. <https://doi.org/10.1109/WCRE.2012.38>
- 1308 [26] Lahari Poddar, Leonardo Neves, William Brendel, Luis Marujo, Sergey Tulyakov,  
1309      and Pradeep Karuturi. 2019. Train One Get One Free: Partially Supervised Neural  
1310      Network for Bug Report Duplicate Detection and Clustering. *arXiv preprint  
1311      arXiv:1903.12431* (2019).
- 1312 [27] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings  
1313      using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).
- 1314 [28] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings  
1315      using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical  
1316      Methods in Natural Language Processing and the 9th International Joint Conference  
1317      on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational  
1318      Linguistics, Hong Kong, China, 3982–3992. <https://doi.org/10.18653/v1/D19-1410>
- 1319 [29] Gerard Salton and Chris Buckley. 1988. Term-weighting approaches in automatic  
1320      text retrieval. *Information Processing & Management* 24, 5 (1988), 513–523. [https://doi.org/10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0)
- 1321 [30] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020.  
1322      DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.  
1323      *arXiv:1910.01108 [cs.CL]* <https://arxiv.org/abs/1910.01108>
- 1324 [31] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its  
1325      application in retrieval. *Journal of Documentation* 28, 1 (1972), 11–21. <https://doi.org/10.1108/eb026526>
- 1326 [32] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more  
1327      accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International  
1328      Conference on Automated Software Engineering (ASE)*. IEEE, 253–262. <https://doi.org/10.1109/ASE.2011.6100061>
- 1329 [33] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo.  
1330      2010. A discriminative model approach for accurate duplicate bug report re-  
1331      trieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software  
1332      Engineering (ICSE)*, Vol. 1. ACM, 45–54. <https://doi.org/10.1145/1806799.1806807>
- 1333 [34] Ting Zhang, Amin Khasteh, Minh-Son Le, Hoan Anh Nguyen, and David Lo. 2023.  
1334      Cupid: Leveraging ChatGPT for More Accurate Duplicate Bug Report Detection.  
1335      *arXiv:2308.10022 [cs.SE]*

Received 16 November 2025