The Journal of Systems and Software 216 (2024) 112109

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Deep semi-supervised learning for recovering traceability links between issues and commits☆

Jianfei Zhu [a], Guanping Xiao [a,*], Zheng Zheng [b], Yulei Sui [c]

[a] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 211106, China
[b] School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, China
[c] School of Computer Science and Engineering, University of New South Wales, Sydney, 2052, Australia

ARTICLE INFO

Dataset link: https://github.com/DSSLink

Keywords:
Traceability links recovery
Deep semi-supervised learning
Self-training
Label propagation
Issue and commit

ABSTRACT

Traceability links between issues and commits record valuable information about the evolutionary history of software projects. Unfortunately, these links are often missing. While deep learning stands as the current state-of-the-art (SOTA) in automated traceability links recovery (TLR), its effectiveness is faced with the practical problem of limited labeled data during training. To overcome this challenge, in this paper, we propose DSSLink, a novel method based on deep semi-supervised learning, enhancing deep-learning-based link recovery tasks. DSSLink first learns knowledge from labeled data through pre-trained model and then leverages deep semi-supervised learning to infer pseudo-labels on unlabeled data. The extended dataset of pseudo-labeled and labeled data re-trains the deep learning model in an iterative process. Our extensive evaluations are conducted on two SOTA traceability methods (T-BERT and BTLink) across four GitHub projects and 11 Apache projects. Specifically, the maximum F1-score improvements for GitHub and Apache projects reached 22.9% and 43.5%, respectively. Evaluation results show that DSSLink is effective in enhancing TLR performance and outperforms TraceFUN, a recent approach that utilizes unlabeled data for TLR. The source code of DSSLink is available at https://github.com/DSSLink.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

1. Introduction

Software development involves creating and maintaining source code in version control systems (e.g., Git, 2023) and submitting issue reports by users and testers in bug tracking systems (e.g., Bugzilla, 2023). The traceability link between issues and commits is crucial for effective software maintenance, as it facilitates developers in performing program comprehension (Liao et al., 2018; Tian et al., 2022), evaluating software quality (Heinemann et al., 2014; Behnamghader et al., 2017), and predicting software defects (Yang et al., 2017; Li et al., 2017). Unfortunately, these traceability links are often unrecorded, making it challenging to maintain software effectively (Bachmann et al., 2010).

To recover missing links between issues and commits, several rule-based and machine learning-based traceability links recovery (TLR) methods have been proposed over the past decade (Bachmann et al., 2010; Wu et al., 2011; Nguyen et al., 2012; Le et al., 2015; Sun et al., 2017b,a). These methods are limited in understanding semantic relationships between issues and commits, hindering the accuracy of link recovery. To address this limitation, researchers have turned to deep learning (Ruan et al., 2019; Lin et al., 2021; Lan et al., 2023), which can effectively capture semantic correlations between terms. However, a significant drawback of deep neural networks is their high reliance on labeled data for training, which can be expensive and time-consuming. It is often impractical to construct large labeled datasets for each deep learning task, making using readily available unlabeled data a promising research direction.

In our previous work, we propose TraceFUN (Zhu et al., 2022), a method to improve TLR performance with unlabeled data. The core idea of TraceFUN is that when an unlabeled artifact shares a similar relationship with a labeled artifact, it is highly probable that it is also connected to the linked objects associated with the labeled artifact. For example, given a link between an issue and a commit, if there is an unlabeled commit with a similar description to the linked commit, they are likely to fix the same issue. This implies that both of them could have links to the issue. The effectiveness of this process is contingent on the abundance of labeled artifacts, as a larger pool increases the likelihood of identifying high-similarity pairs. However, when faced

☆ Editor: Alexander Chatzigeorgiou.
* Corresponding author.
E-mail address: gpxiao@nuaa.edu.cn (G. Xiao).

https://doi.org/10.1016/j.jss.2024.112109
Received 26 June 2023; Received in revised form 11 December 2023; Accepted 20 May 2024
Available online 24 May 2024

with a limited of labeled artifacts, the retrieval of high-similarity pairs becomes challenging. Consequently, creating high-quality new links becomes difficult, rendering TraceFUN ineffective in such scenarios.

In real-world projects, the high cost of labeled data often results in only small labeled datasets being available while large amounts of unlabeled data exist. Deep neural networks have demonstrated superior performance on supervised tasks such as image classification when large labeled datasets are available. Considering the small amount of labeled data in real-world training datasets, there is increasing research interest in applying semi-supervised learning to deep neural networks, namely deep semi-supervised learning (DSSL). It has achieved good results in the field of computer vision (Lee et al., 2013; Tarvainen and Valpola, 2017; Miyato et al., 2018; Xie et al., 2020; Berthelot et al., 2019b,a; Sohn et al., 2020; Zhang et al., 2021; Zheng et al., 2022) and natural language processing (Chen et al., 2020; Dong et al., 2023). Deep semi-supervised learning is a method of using limited labeled and large amounts of unlabeled data to train deep neural networks, which can effectively overcome the limitation of TraceFUN.

In this paper, we propose DSSLink, a deep semi-supervised learning method for recovering links between issues and commits. The method initiates by pre-training an initial model (T-BERT (Lin et al., 2021) and BTLink (Lan et al., 2023)) using limited labeled data. Then, unlabeled software artifacts are paired based on temporal rules. In the third step, pseudo-labels for unlabeled artifact pairs are inferred through deep semi-supervised learning, i.e., self-training (Lee et al., 2013) and label propagation (Iscen et al., 2019). Following this, pseudo-labeled data is selected in the fourth step, and a class-balanced dataset is constructed. The fifth step involves re-training the model using both pseudo-labeled and labeled data. This process is iterated from steps three to five, resulting in a deep neural network learned from labeled and unlabeled data, effectively enhancing TLR performance. We evaluate DSSLink using issue and commit data collected from 15 open-source software projects, including four GitHub projects (Flask, Pgcli, Keras, and Scrapy) and 11 Apache projects (ant-ivy, Avro, Beam, Buildr, Giraph, Isis, logging-log4net, Nutch, OODT, Tez, and Tika).

In summary, this paper makes the following key contributions:

- We propose DSSLink, a approach based on deep semi-supervised learning that recovers traceability links between issues and commits using labeled and unlabeled data. DSSLink is publicly available as an open-source project: https://github.com/DSSLink.
- We use fifteen open-source software projects to evaluate the effectiveness of DSSLink using unlabeled data with different amounts of labeled data. DSSLink significantly improves the performance of the two state-of-the-art (SOTA) methods, i.e., T-BERT (Lin et al., 2021) and BTLink (Lan et al., 2023), and outperforms TraceFUN, a recent approach that utilizes unlabeled data for TLR.
- We empirically evaluate whether the proportion of labeled and unlabeled data and the pairing rule of unlabeled data have an impact on the performance of DSSLink. Based on our analysis, we provide practical guidelines for using unlabeled data with DSSLink in recovering trace links between issues and commits.

The remainder of the paper is structured as follows. Section 2 briefly introduces background concepts related to traceability links recovery between issues and commits and deep semi-supervised learning techniques. Section 3 describes our DSSLink method. Section 4 presents the experimental setup and evaluation of DSSLink. Section 5 reports our experimental results, which are further analyzed and discussed in Section 6. Section 7 lists the main threats to the validity. Section 8 presents related work. Finally, Section 9 concludes the paper.

2. Background concepts

In this section, we introduce the fundamental concepts related to the main contributions of this paper.

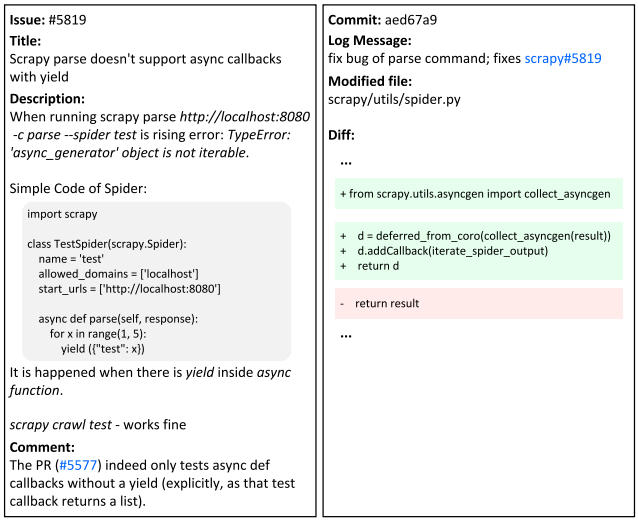

Fig. 1. Issue #5819 and its corresponding commit aed67a9 from Scrapy project.

2.1. Trace links between issues and commits

Issues and commits are common software artifacts that play a significant role in software development. Traceability links, which establish connections between issues and commits, are crucial for software engineering research and addressing practical problems (Tian et al., 2021). For example, various issues may arise throughout the software life cycle, which users or testers report through bug reports to describe and request fixes. Upon receiving the issue report, the development team addresses the issue and documents the fix in the code commit, usually referencing the corresponding issue report ID. The user or tester then verifies the resolution and closes the issue report. This process establishes a link between the issue and the commit through identifiers, as depicted in Fig. 1, which shows an example from the Scrapy project. Issue (#5819) reports the problem: *Scrapy parse does not support async callbacks with yield.* The commit (aed67a9) resolves the issue by referencing the issue ID in the commit message. However, due to various reasons, developers may inadvertently omit identifiers, leading to missing links between issues and commits. To recover these missing links, researchers have proposed several automated traceability links recovery (TLR) methods, e.g., FRLink (Sun et al., 2017b), DeepLink (Ruan et al., 2019), and Hybrid-Linker (Mazrae et al., 2021).

In this paper, we formulate the traceability links recovery problem between issues and commits as follows: given two sets of software artifacts, denoted as s for issues and t for commits. Partial traceability links between issues and commits have been obtained, while the remaining software artifacts are considered as unlabeled data. Among them, there are a total of l labeled trace links, represented as $X_l = \{(s,t)_i \mid i \in (1,l)\}$. The labels for the pairs of issues and commits with a trace link are assigned as 1, while the pairs without a trace link are labeled as 0, denoted as $y_l \in \{0,1\}$. The unlabeled data consists of u pairs and the total number of artifact pairs is $n = l + u$. The unlabeled data can be represented as $X_u = \{(s,t)_j \mid j \in (l+1,n)\}$. The labels y_u for these pairs are unknown and assigned as -1.

2.2. Traceability links recovery method

Fig. 2 illustrates two current SOTA TLR methods, T-BERT blue (Lin et al., 2021) and BTLink (Lan et al., 2023), which both contain BERT embedding layer, fusion layer, and classifier layer. In the BERT embedding layer, software artifacts are represented as feature vectors. This layer can use different BERT models and different model structures, and we applied the best model structure of these two methods. Specifically,

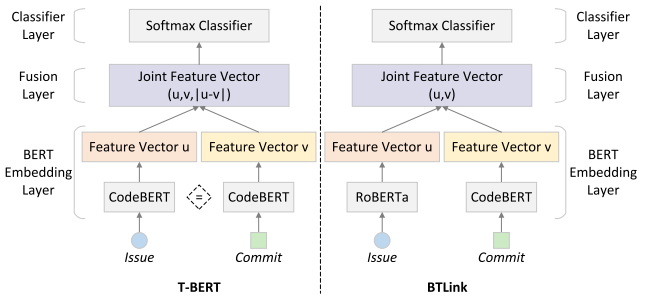

Fig. 2. The structure of BERT-based traceability methods. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

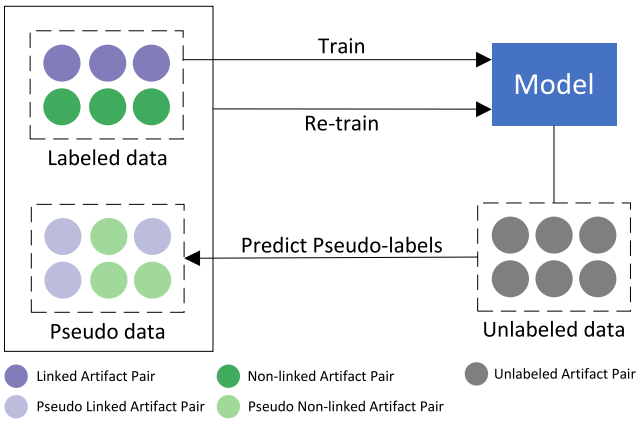

Fig. 3. Illustration of the self-training process.

the T-BERT method uses the same CodeBERT model to receive input from software artifacts, while BTLink uses RoBERTa and CodeBERT to receive input from issues and commits, respectively.

In the fusion layer, BTLink splices the feature vectors u, v into a joint feature vector. On this basis, T-BERT adds a bitwise difference vector $|u - v|$ between the two vectors. Finally, the joint feature vector enters the classifier layer, a softmax function, to produce a confidence score representing the likelihood of a traceability link between the two artifacts.

We employ deep semi-supervised learning to train the TLR model using a combination of labeled and unlabeled data. We denote the deep neural network as f_θ, where θ is the network parameter. The network performs feature representation on the input X_i, and constructs the descriptor of the ith sample using $\phi_\theta((s,t)_i) = v_i$. Subsequently, a softmax function is applied, and the model output for the ith sample is denoted as $\hat{y}_i = arg\ \max_j f_\theta((s,t)_i)_j$, where the subscript j represents the jth dimension of the vector.

2.3. Deep semi-supervised learning

Deep neural networks possess powerful representation learning capabilities, effectively capturing feature representations of data and demonstrating robust generalization to unknown data after extensive training on large-scale datasets. However, the challenge of insufficient labeled data is pervasive in current deep-learning tasks. To address such problems, semi-supervised learning has been introduced into deep learning (Lee et al., 2013; Xie et al., 2020; Iscen et al., 2019). Semi-supervised learning excels in learning from unlabeled data, exhibiting stronger learning capabilities in tasks where labeled data is limited. The amalgamation of semi-supervised learning and deep learning results in deep semi-supervised learning, aiming to train a model with robust generalization capabilities by combining a small amount of human-expert-labeled data with a large amount of unlabeled data.

To learn the data distribution relationship, there are three underlying assumptions in semi-supervised learning:

1. Smoothing assumption (Belkin et al., 2006): close data points tend to share the same label. This assumption implies that if two samples reside in a high-density region and are close to each other, there is a higher probability that they belong to the same class.
2. Clustering assumption (Chapelle et al., 2002): data points tend to form discrete clusters. According to this assumption, if two samples are part of the same cluster, they are likely to have the same class label.
3. Manifold assumption (Belkin et al., 2006): if two samples are close to each other in high-dimensional space, they are also likely to be close to each other in low-dimensional manifolds. This assumption implies that it is possible to embed high-dimensional data into a low-dimensional manifold where the data's inherent structure is preserved. In this low-dimensional manifold, samples located in the same local neighborhood are expected to have similar class labels.

There are two primary learning paradigms in semi-supervised learning (Chapelle et al., 2006):

1. Inductive learning (Hayes et al., 2010): it aims to learn general patterns from a training set to make predictions on unknown data. Emphasizing generalization to new, unseen data takes precedence over specific examples in the training dataset. The self-training algorithm is a prominent application within this semi-supervised paradigm.
2. Transductive learning (Arnold et al., 2007): it aims to assign a label to every sample in the training set. The focus is on maximizing the prediction accuracy of unlabeled samples within the training set by extracting detailed information from the existing data. The label propagation algorithm is a prominent application within this semi-supervised paradigm.

Deep semi-supervised learning has yielded notable research achievements (Yang et al., 2022), and the above two paradigms and their corresponding semi-supervised learning algorithm applications, i.e., self-training and label propagation, have also been introduced into the field of deep learning. Notably, Lee et al. (2013) applied self-training to deep learning tasks, while Iscen et al. (2019) applied label propagation to the deep learning. These works have become two representative deep semi-supervised learning studies. Therefore, we select self-training and label propagation as our semi-supervised learning algorithms based on the aforementioned two paradigms, and we iteratively train deep neural networks through semi-supervised learning in deep learning tasks, following the work from Lee et al. (2013) and Iscen et al. (2019). Detailed descriptions of these two deep semi-supervised learning algorithms are presented below.

2.3.1. Self-training

As depicted in Fig. 3, self-training mainly includes the following four key steps:

1. Train an initial classifier utilizing labeled data. In DSSLink, we use SOTA methods T-BERT and BTLink as classifiers.
2. Employ the inception classifier to classify the unlabeled data and assign pseudo-labels to them.
3. Apply a specific rule to select pseudo-labeled data with a higher accuracy rate.
4. Combine the selected pseudo-labeled data with labeled data and retrain the classifier.

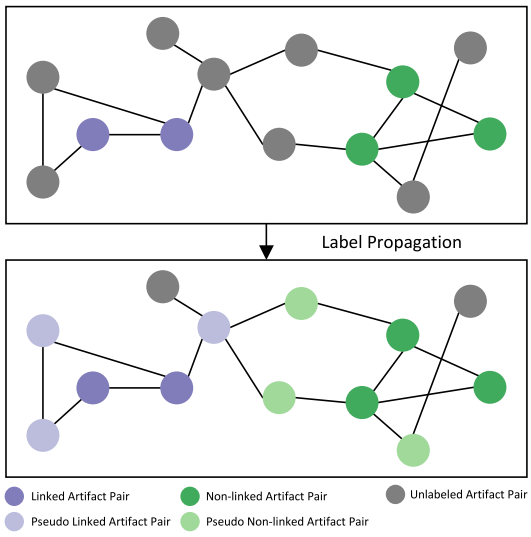

Fig. 4. Illustration of the label propagation process.

For the link recovery task, the objective of the classifier is to determine whether a trace link exists between a given pair of issues and commits. In DSSLink, the initial model (i.e., T-BERT and BTLink) is pretrained using partial trace links. Then, this trained model is used to predict pseudo-labels for unlabeled artifact pairs. The one-hot encoded labels $\hat{y_{ij}}$ for an artifact pair $(s,t)_i$ is defined as follows:

$$\hat{y_{ij}} = \begin{cases} true, & \text{if } j = arg\max_{j'} f_\theta((s,t)_i)_{j'} \\ false, & \text{otherwise} \end{cases} \tag{1}$$

where $f_\theta((s,t)_i)_{j'}$ represents the predicted probability of class j for the artifact pair $(s,t)_i$, as determined by the model parameterized by θ. If the class index corresponding to the highest predicted probability is j, the class j of sample i is assigned as *true*; otherwise, it is assigned as $false$. For the TLR task between issues and commits, there are two classes, the presence of a trace link and the absence of a trace link. The pseudo-label of sample $\hat{y}_i$ is expressed by the following:

$$\hat{y}_i = \begin{cases} 1, & \text{presence of a trace link} \\ 0, & \text{absence of a trace link} \end{cases} \tag{2}$$

The pseudo-labeled data is not completely accurate, and false pseudo-labels will mislead model learning. Therefore, in self-training, it is crucial to choose correctly classified pseudo-label samples based on certain selection criteria. A commonly used selection method is to filter predictions for which the model lacks confidence through a predetermined confidence threshold (Lee et al., 2013; Xie et al., 2020; Sohn et al., 2020). Predictions exceeding the threshold and demonstrating higher confidence are selected for further training. In DSSLink, a confidence threshold (0.8) is utilized to filter out the predictions for which the model lacks confidence. The filtered predictions, denoted as X'_u, consist of artifact pairs $((s,t)_i, \hat{y}_i)$, where $\hat{y}_i$ represents the pseudo-label predicted by the model for the pair. The selection is based on the condition that the maximum predicted probability $max(f_\theta((s,t)_i))$ surpasses the specified threshold.

$$X'_u = \left\{ \left((s,t)_i, \hat{y}_i\right) \mid max\left(f_\theta((s,t)_i)\right) > threshold \right\} \tag{3}$$

Finally, the labeled dataset X_l and the filtered pseudo-labeled dataset X'_u are combined. The TLR model is then retrained using the combined dataset.

2.3.2. Label propagation

Label propagation is a type of graph-based semi-supervised learning method. As shown in Fig. 4, the graph's edges represent similarities between data points, allowing for the propagation of labels from labeled data to unlabeled data. To begin the label propagation process, we use a pre-trained model to obtain the feature representation of both the labeled and unlabeled data, forming the descriptor set $V = (v_1, \ldots, v_l, v_{l+1}, \ldots, v_n)$. Then, based on the descriptor set V, the similarity k-adjacency graph of the data is constructed. To construct a sparse affinity matrix $A \in \mathbb{R}^{n\times n}$, the elements are computed as follows:

$$a_{ij} = \begin{cases} \left[v_i^T \cdot v_j\right], & \text{if } i \neq j \wedge v_i \in \mathrm{NN_k}\left(v_j\right) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

In Eq. (4), $\mathrm{NN_k}\left(v_j\right)$ denotes the set of the k neatest neighbors of v_j in X. The weight matrix W of the similarity k-adjacent graph is then constructed as as $W = A + A^T$, and its symmetrically normalized counterpart is calculated as $W = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$, where $D = \mathrm{diag}\left(W\mathbf{1}_n\right)$ is the degree matrix, and 1_n represents an all-ones vector of size n.

Next, we proceed to construct a label matrix $Y \in \mathbb{R}^{n\times 2}$. The elements of Y are calculated as follows:

$$Y_{ij} = \begin{cases} 1, & \text{if } i \in X_l \wedge y_i = j \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

where X_l represents the labeled data and y_i denotes the label of sample i. The label information is then propagated from the labeled data X_l to unlabeled data X_u in the feature space. This is achieved by calculating the diffused matrix Z using the following formula:

$$Z = (I - \alpha W)^{-1} Y \tag{6}$$

Here, $\alpha \in [0, 1)$ is a hyper-parameter. Finally, for an unlabeled example x_i, the probability that a traceability link exists between its corresponding artifacts is determined as:

$$\hat{y}_i = arg\max_j z_{ij} \tag{7}$$

In Eq. (7), z_{ij} is the (i,j) element of matrix Z. To account for the uncertainty in the probability obtained from Z, each row of Z is normalized. This normalization ensures that each row represents the probability of a traceability link's existence for a particular sample.

The pseudo-labeled data in label propagation is also not completely accurate. However, unlike self-training, transductive learning aims to assign a label to each sample in the training data. Consequently, each pseudo-labeled sample is assigned a weight that reflects the certainty of the prediction (Iscen et al., 2019). We use entropy to measure the uncertainty of the predicted probability. The higher the uncertainty of the sample, the lesser its influence on the loss calculation. We mitigate the negative impact of false pseudo-labels on the model by calculating weights. The weight ω of the unlabeled data is defined as follows:

$$\omega_i = 1 - \frac{H(\hat{z}_i)}{\log_{,(c)}} \tag{8}$$

where $\hat{z}_i$ is the ith row of Z after normalization, function H is the entropy function, and c is the number of classes.

3. Methodology

3.1. Challenges and overview

To leverage unlabeled data through deep semi-supervised learning for TLR, we need to address the following two main challenges (C).

C1: How to pair unlabeled software artifacts to overcome the issue of class imbalance? In the task of TLR, the objective is to predict the presence of trace links between pairs of software artifacts (i.e., issues and commits). The labeled data consists of pairs of artifacts for which the link status is known (i.e., a trace link exists), while the unlabeled data comprises pairs where the link status is unknown. Typically, to create the pairs of software artifacts, a common approach is to calculate the Cartesian product of the two sets. For example, there are 427 issues and 432 commits in the collected Flask project, thereby

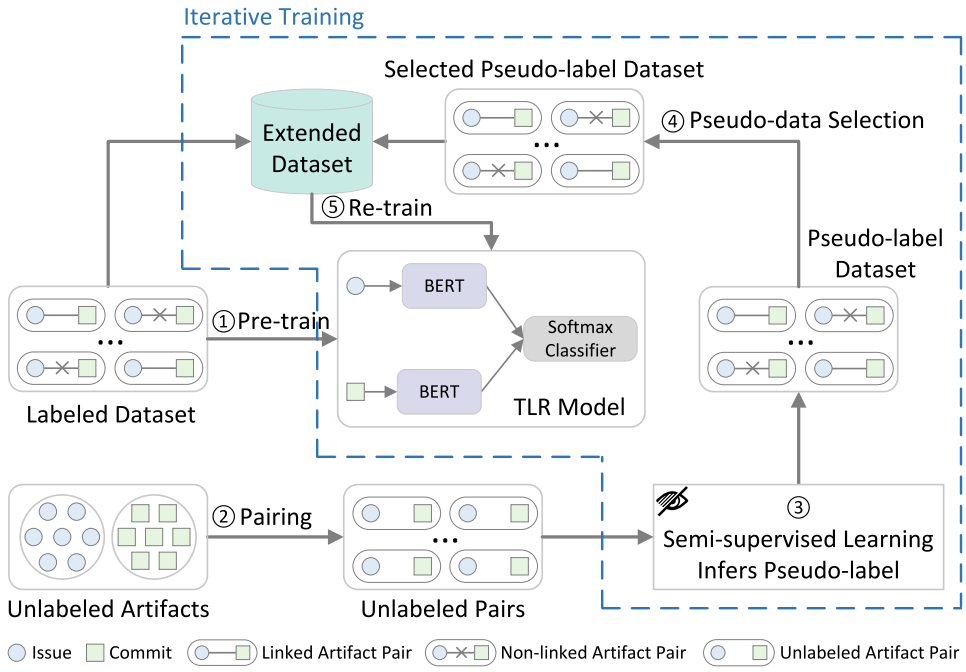

Fig. 5. Overview of DSSLink.

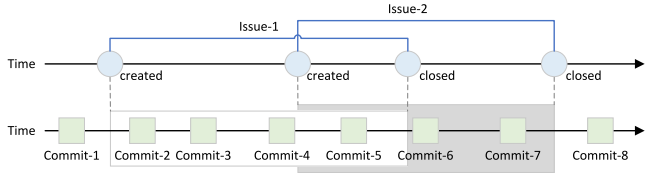

Fig. 6. Illustration of time relationship between issues and commits.

the Cartesian product generating 184,464 software artifact pairs (427 * 432).

However, this approach often leads to a severe class imbalance issue. In many software projects, the number of artifact pairs without trace links far exceeds the number of pairs with links. In the Flask project, out of all the pairs of artifacts, only 432 pairs have labeled trace links, while the remaining 184,032 pairs are unlabeled. This indicates a substantial class imbalance, as the number of labeled pairs is significantly smaller compared to the number of unlabeled pairs. The majority of the unlabeled pairs are expected to be without trace links. This class imbalance poses challenges for training a reliable TLR model. The model can easily become biased towards the majority class (i.e., the true label is 0), leading to poor performance in predicting trace links.

C2: How to select pseudo-data to generate class balance dataset? After inference of pseudo-labels for the unlabeled data through semi-supervised learning, the majority of the pseudo-labels for the unlabeled artifact pairs are inferred as negative sample, leading to a class imbalance issue.

To address the two challenges, this section introduces our DSSLink, which consists of five phases: (1) pre-training, (2) unlabeled artifact pairing, (3) pseudo-label inference through semi-supervised learning, (4) pseudo-data selection, and (5) iterative retraining, as depicted in Fig. 5.

3.2. Pre-training

In the first phase of DSSLink, we employ supervised training to train a initial model using a limited set of labeled software artifact pairs, denoted as X_l. This process enables the model to learn latent features for software artifact data. After pre-training, the initial model is capable of predicting labels and providing feature representation for unlabeled data, which further serve as valuable inputs for subsequent self-training and label propagation processes.

Note that the classifier used in the pre-training stage is determined by the TLR method to be enhanced. In this paper, we enhance T-BERT (Lin et al., 2021) and BTLink (Lan et al., 2023), respectively. Therefore, these two classifiers are selected as the initial models.

3.3. Unlabeled artifacts pairing

To address the class imbalance problem arising in pairing unlabeled software artifacts (C1), we propose a novel pairing rule aimed at reducing the number of artifact pairs. Inspired by the work of Wu et al. (2011), which focuses on recovering missing links through feature learning from explicit links, we leverage the temporal characteristics of issues and commits. Specifically, we take into account the time interval between the creation of an issue report and the submission of the corresponding code fix. In the standard software development process, developers typically modify and submit the code after creating an issue report, followed by closing the report. Hence, the submission time of the code falls within the timeframe between the issue report's creation and closing. Based on this insight, we introduce a new pairing rule: each issue is paired with all commits that are submitted within its creation and closing time window. Note that the time-based approach is only applied to the closed issues. This means that open (re-opened) issues would be excluded from the pairing process.

As illustrated in Fig. 6, we can observe two issues and eight commits. The Cartesian product approach would generate 16 software artifact pairs. However, by examining the temporal relationship between issues and commits, we can identify specific pairs that fall within the respective creation and closing time window. For instance, commits No. 2, No. 3, No. 4, and No. 5 fall within the creation and closing time of Issue-1, while commits No. 5, No. 6, and No. 7 fall within the creation and closing time of Issue-2. Therefore, we pair Issue-1 with Commit-2, Commit-3, Commit-4 and Commit-5, and Issue-2 with Commit-5, Commit-6, and Commit-7. By adhering to the time-based relationship, we generate only seven artifact pairs, effectively reducing the number of pairs compared to the Cartesian product method. For the

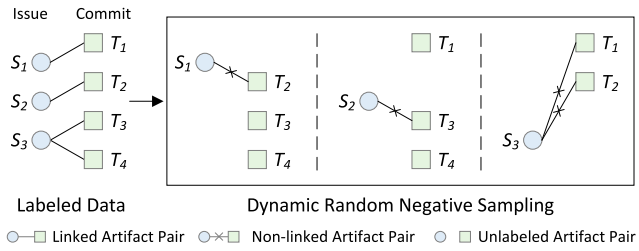

Fig. 7. Illustration of the DRNS process.

aforementioned Flask dataset, the proposed pairing rule results in the generation of only 1885 artifact pairs. This is a substantial reduction compared to the 184,032 pairs obtained by using the Cartesian product pairing method, leading to a significant decrease in the number of unlabeled data instances with a true label of 0.

3.4. Semi-supervised learning

DSSLink employs a powerful semi-supervised learning method (i.e., self-training or label propagation) at the beginning of each iteration. This method utilizes the capabilities of deep neural networks to infer pseudo-labels for unlabeled pairs. By leveraging local features extracted from labeled data and the sample distribution of unlabeled data, the semi-supervised learning method estimates the presence or absence of traceability links between the unlabeled issues and commits. The inferred outcomes from this process serve as pseudo-labels for the pairs of unlabeled artifacts in subsequent training steps.

3.5. Pseudo-data selection

To generate a class balance dataset from the inferred pseudo-data (C2), DSSLink adopts dynamic random negative sampling (DRNS) introduced by Lin et al. (2021). Suppose an issue s_i that is connected to n commits through trace links, to address the class imbalance problem, DRNS involves removing a certain number n of commits linked to s_i from the set T that contains all the commits. Then, n commits are randomly selected from the remaining set T and paired with issue s_i as negative samples. This sampling ensures that the number of negative samples matches the number of positive samples.

To generate negative samples, DRNS does not need to consider any specific attributes (code or text properties) of artifacts. As shown in Fig. 7, issue S_1 links to commit T_1. To generate negative samples for issue S_1, DRNS removes T_1 from the commit set (T_1, T_2, T_3, T_4) and then randomly selects one commit (e.g., T_2) from the remaining set (T_2, T_3, T_4). Because only one labeled link is connected to issue S_1, DRNS only generates one negative sample for S_1, i.e., (S_1, T_2). The same process applies to issues S_2 and S_3. Therefore, negative samples (S_2, T_3), (S_3, T_1), and (S_3, T_2) will be created for issues S_2 and S_3, respectively.

To construct a class-balanced dataset, we select instances with a pseudo-label of 1, i.e., the unlabeled artifact pair that is inferred as a positive sample through deep semi-supervised learning. The same number of negative samples is then generated for these data using DRNS. This process ensures a balanced representation of positive and negative samples within the pseudo-label dataset.

3.6. Iterative training

After obtaining a set of pseudo-labeled data through semi-supervised learning, DSSLink iteratively trains the TLR model using both labeled and pseudo-labeled data. At the beginning of each epoch, the pre-trained model is utilized to infer pseudo-labels for the unlabeled data. These pseudo-labels serve as additional information to guide the training process. Subsequently, the model is retrained using a combination of the pseudo-labeled and labeled data. This combined dataset is used to update the model's parameters, marking the completion of the current epoch.

This iterative process continues, with the model being updated at the end of each epoch based on the combined dataset. During the iterative training process in DSSLink, both labeled and unlabeled data are utilized for training in a supervised manner. Separate loss terms are computed for the labeled and unlabeled data, and a combined loss is computed. Specifically, the labeled loss is defined as:

$$loss_l = \frac{1}{|X_l|} \sum_{s,t,y \in X_l} H\left(y, f_\theta(s,t)\right) \tag{9}$$

where $H(p, q)$ is the cross-entropy between distributions p and q.

In self-training, only pseudo-labeled data whose predicted probability exceeds the confidence threshold will contribute to the loss calculation. The threshold is related to the quality of the selected pseudo-labeled data. Through preliminary experiments, we fine-tuned the confidence threshold, discovering that setting it to 0.8 yields better final performance. This choice aligns with the work of Xie et al. (2020), where a confidence threshold of 0.8 was also employed. Thus, based on our preliminary experimental findings and drawing from existing research, we define the confidence threshold in self-training as 0.8 to maximize the inclusion of true links. The unlabeled data loss of self-training is computed as:

$$loss_u = \frac{1}{|X'_u|} \sum_{s,t,\hat{y} \in X'_u} H\left(\hat{y}, f_\theta(s,t)\right) \tag{10}$$

In label propagation, each pseudo-labeled data participates in the loss calculation, but their loss is weighted by ω. Therefore, the unlabeled data loss of label propagation is computed as:

$$loss_u = \frac{1}{|X_u|} \sum_{s,t,\hat{y} \in X_u} H\left(\hat{y}, f_\theta(s,t)\right) * \omega \tag{11}$$

Finally, the combined loss is calculated as the sum of the labeled loss and a scaled version of the unlabeled data loss. The scaling factor λ_u is a hyper-parameter controlling the contribution of the unlabeled data loss:

$$loss = loss_l + \lambda_u loss_u \tag{12}$$

4. Experimental setup and evaluation

4.1. Evaluation datasets

To evaluate the effectiveness of DSSLink, we utilized datasets from fifteen open-source software projects, including four GitHub projects and 11 Apache projects. Among them, three GitHub projects (Flask, Pgcli, and Keras) are obtained from T-BERT (Lin et al., 2021), and the Scrapy project is collected by us, following the same data collection rules as Lin et al. (2021). To ensure a robust evaluation, we employed a 5-fold stratified cross-validation (Wong and Yeh, 2019). For each project, the labeled dataset was divided into five folds, with four folds used for training and the remaining fold reserved for testing. To facilitate model training, we applied a validation split ratio of 0.2, i.e., 20% of the training set was allocated for validation purposes. Additionally, 11 Apache projects (ant-ivy, Avro, Beam, Buildr, Giraph, Isis, logging-log4net, Nutch, OODT, Tez, and Tika) are obtained from Lan et al. (2023). The training, validation, and test sets of these projects have been divided by them, as shown in Table 1.

To generate labeled and unlabeled datasets for training, we divided the training set into two parts. One part was designated as labeled data, while the other part was anonymized and treated as unlabeled data. We varied the split ratio to generate datasets from Github projects with ratios of 10%, 30%, 50% and 70%. Additionally, we included an extreme case where only 10 instances were labeled. To further verify the generalizability of DSSLink, we conducted comparison experiments

Table 1
Collected datasets for DSSLink evaluation.

Dataset	Train			Validation			Test		
	Issue	Commit	Link	Issue	Commit	Link	Issue	Commit	Link
Flask	474	481	481	120	121	121	150	150	150
Pgcli	334	338	338	85	85	85	105	106	106
Keras	352	353	353	89	89	89	110	110	110
Scrapy	467	481	481	119	121	121	149	150	150
Avro	2031	2077	2011	1265	1329	245	1287	1329	286
Beam	7161	7497	718	7198	7510	748	7226	7538	734
Buildr	526	814	667	314	418	75	314	417	87
Giraph	803	691	559	451	412	63	450	420	84
Ivy	945	1110	921	493	548	120	487	564	117
Isis	2148	10,821	8360	1995	6042	1070	1970	6043	1045
log4net	195	223	190	80	80	29	72	79	29
Nutch	1747	1818	1440	829	887	183	839	903	169
OODT	701	1194	1015	457	569	116	438	582	123
Tez	2678	2667	2204	2123	2157	288	2143	2162	293
Tika	2207	3467	2510	1460	1856	295	1435	1860	358

Table 2
Newly collected datasets for DSSLink evaluation.

Dataset	Issue	Commit	Time frame
Flask	243	244	Up to 2023-03-31
Pgcli	63	63	Up to 2023-03-31
Keras	242	255	Up to 2023-03-31
Scrapy	1623	1786	Up to 2023-04-19

with BTLink on the Apache projects, using a 10% split ratio of labeled data.

Furthermore, we collected the most recent data from the Flask, Pgcli, and Keras projects as unlabeled data, adhering to the data collection guidelines specified by Lin et al. The data was collected up to March 31, 2023. Therefore, the unlabeled dataset consists of anonymized data from the training set and newly collected data.

Moreover, we collected data from the Scrapy project as a new project to evaluate the performance of DSSLink. The data collection deadline for this project was set to April 19, 2023. We labeled this data using the same quantity of labeled data as the Flask project, and the same processing was applied as for the three aforementioned projects.

Table 2 presents the newly collected software artifacts and their respective collection deadlines. Note that the newly collected data of the Flask, Pgcli, and Keras projects do not overlap with the software artifacts provided in the work conducted by Lin et al. (2021). Table 3 shows the average counts of labeled and unlabeled software artifacts for the different scaled datasets of the four GitHub projects. These data will be used in research question 3.

4.2. Implementation details

Experiment Environments. The development environment of DSSLink is as follows: Python 3.8, PyTorch 1.5.0. The enhanced experiments on T-BERT are conducted on a workstation with two Intel Xeon Gold 6230R CPUs @ 2.10 GHz (26 cores with 52 threads), 160 GB memory, 256 GB SSD, 8 TB HDD storage, and three Nvidia RTX 2080Ti GPU cards. Because of the architecture of BTLink, it requires higher computing resources compared to T-BERT, e.g., more GPU memory. We used the same GPU configuration in Lan et al. (2023). The original BTLink and self-training models were trained on a machine equipped with Nvidia RTX 3090. In addition, label propagation requires more graphic memory for graph operations. Thus, the label propagation model for BTLink was trained on a machine equipped with Nvidia A100-PCIE-40 GB.

Settings for Baseline T-BERT. The training parameters for the T-BERT, which utilizes only labeled data, are as follows: the number of epochs is 200; batch size is 4; learning rate is 0.00004; network architecture is faster SIAMESE.

Settings for Baseline BTLink. The training parameters for the BTLink, which utilizes only labeled data, are as follows: the number of epochs is 20; batch size is 4; learning rate is 0.00005.

***Settings for* DSSLink.** In DSSLink, the traceability model training, which incorporates both labeled and unlabeled data, consists of two stages: pre-training and DSSL iterative training. For T-BERT, the training parameters of pre-training are as follows: the number of epochs is 100; batch size is 4; learning rate is 0.00004; network architecture is faster SIAMESE. The iterative training parameters of deep semi-supervised learning are as follows: the number of epochs is 100; labeled data batch size is 1; unlabeled data batch size is 3; learning rate is 0.00004; network architecture is faster SIAMESE. For BTLink, the training parameters of pre-training are as follows: the number of epochs is 10; batch size is 4; learning rate is 0.00005. The iterative training parameters of deep semi-supervised learning are as follows: the number of epochs is 10; labeled data batch size is 1; unlabeled data batch size is 3; learning rate is 0.00005.

Note that the maximum batch size is set to 4 due to the limited computing power of the GPU. In deep semi-supervised learning, it is recommended to have a larger amount of unlabeled data compared to the labeled data (Sohn et al., 2020; Zhang et al., 2021; Chen et al., 2020). Therefore, we set the batch size for labeled data as 1 and the batch size for unlabeled data as 3.

Settings for TraceFUN. Since TraceFUN generates new pseudo-label links based on a percentage of the original labeled data, we adopt the percentage setting that yielded the highest TLR performance improvement in our previous work (Zhu et al., 2022). Specifically, the best percentage setting for Flask, Pgcli, and Keras are 110%, 110%, and 5%, respectively. The training parameters for TraceFUN are as follows: the number of epochs is 200; batch size is 4; learning rate is 0.00004; network architecture is faster SIAMESE.

4.3. Evaluation metrics

We used Precision, Recall, F1-score, F2-score, and Mean Average Precision (MAP) as evaluation metrics.

Precision: Precision measures the accuracy of a model's positive predictions by calculating the proportion of true positives among all instances predicted as positive, and it is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

Recall: Recall evaluates the model's ability to capture all actual positive instances by measuring the proportion of true positives among all true positive instances and false negatives, and it is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

where TP represents the instances correctly predicted as positive by the model; TN represents instances correctly predicted as negative by the model; FP represents instances incorrectly predicted as positive by the model; FN represents instances incorrectly predicted as negative by the model.

F-scores: F-scores represent the harmonic mean of precision and recall. It is calculated using the formula:

$$F_\beta = \frac{(1 + \beta^2) * precision * recall}{\beta^2 * precision + recall} \tag{15}$$

where β is a positive real factor that determines the relative importance of recall compared to precision. F1-score assigns equal weight to precision and recall, corresponding to $\beta = 1$. F2-score prioritizes recall by assigning twice the weight to recall compared to precision, corresponding to $\beta = 2$.

Mean Average Precision (MAP): Average precision (AveP) is the average of the maximum precision values achieved at different recall levels. For each issue S_i, $AveP$ is computed based on the position of all relevant commits T in the ranking. MAP is then calculated as the

Table 3
The amount of labeled and unlabeled artifacts in Flask, Pgcli, Keras and Scrapy datasets.

Dataset	Issue (labeled/unlabeled)					Commit (labeled/unlabeled)				
	10	10%	30%	50%	70%	10	10%	30%	50%	70%
Flask	10/707	48/669	143/576	239/481	333/387	10/715	48/677	144/581	240/485	336/389
Pgcli	10/387	33/365	101/298	168/231	233/165	10/391	33/368	101/300	169/232	236/165
Keras	10/584	35/559	105/489	176/418	246/348	10/598	35/573	105/503	176/432	247/361
Scrapy	10/1362	47/1325	142/1234	235/1141	328/1047	10/1505	48/1467	144/1371	240/1275	336/1179

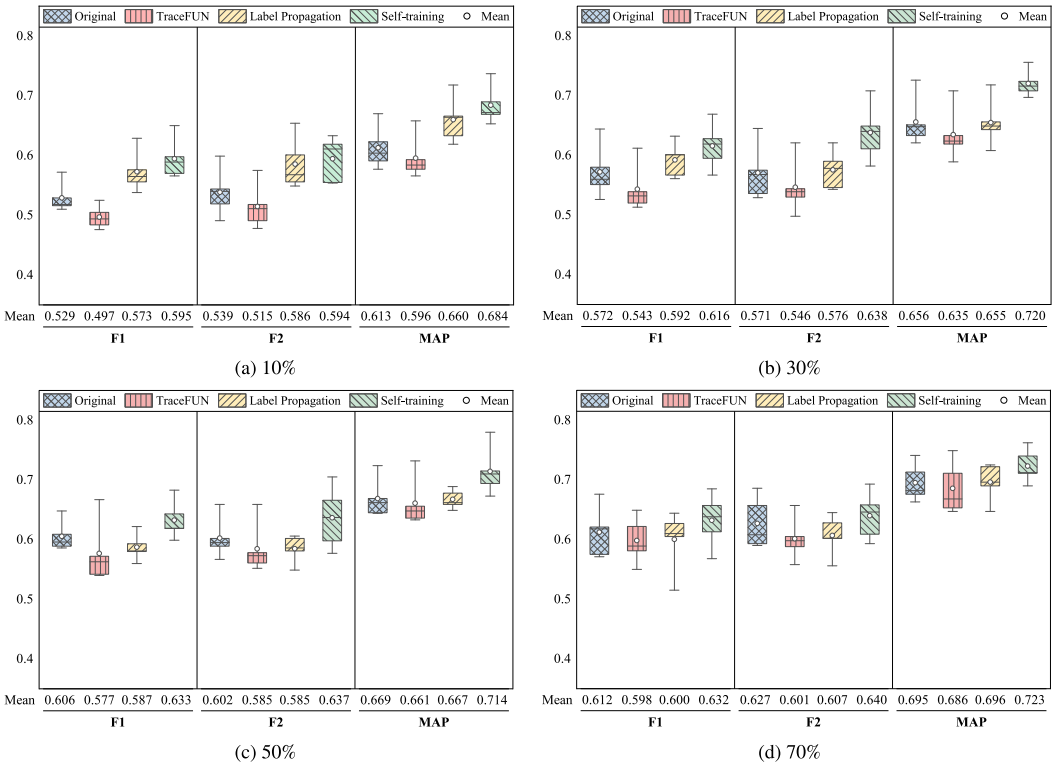

Fig. 8. The evaluation results of Flask for original (T-BERT), TraceFUN, and DSSLink (label propagation and self-training).

average of the AveP values. In our evaluation, we used MAP@3, where only the top 3 ranked artifacts ($rank_i$) contribute to $AveP$:

$$AveP@3 = \frac{\sum_i^n P}{n}, P = \begin{cases} P@i, & \text{if } rank_i \leqslant 3 \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

$$MAP@3 = \frac{1}{Q}\sum_{q=1}^{Q} AveP@3 \tag{17}$$

4.4. Research questions

Our research aims to answer the following five Research Questions (RQs):

- **RQ1:** Can DSSLink enhance the performance of TLR when the amount of labeled data is limited?
- **RQ2:** How does DSSLink compare to TraceFUN in improving the performance of TLR when the amount of labeled data is limited?
- **RQ3:** What is the impact of the ratio between labeled and unlabeled data on the performance of DSSLink?
- **RQ4:** What is the impact of the pairing rule of unlabeled data on the performance of DSSLink?
- **RQ5:** How significantly does DSSLink impact the computational efficiency of traceability methods?

5. Results

In this section, we report the experiment results of our five RQs.

5.1. RQ1: Can DSSLink enhance the performance of TLR when the amount of labeled data is limited?

Figs. 8, 9, and 10 depict the evaluation results for the Flask, Pgcli, and Keras projects, respectively. Each project's result graph presents four boxplots representing the evaluation results for datasets with labeled data volumes of 10%, 30%, 50%, and 70%. The boxplots illustrate three evaluation metrics: F1-score, F2-score, and MAP. Additionally, each evaluation metric includes four models: original (T-BERT model trained with labeled data only), TraceFUN (TLR model trained using the TraceFUN method), label propagation and self-training (T-BERT model trained with DSSLink using different semi-supervised learning algorithms as described in Section 3). The mean values of the three evaluation metrics are also provided in the figures.

It can be observed from Figs. 8 to 10 that DSSLink demonstrates improved performance compared to the original model in most cases. For example, in Fig. 8(a), the evaluation metrics (F1-score, F2-score, and MAP) of label propagation and self-training are higher than those of the original model. For label propagation, there is an improvement of 0.044, 0.047, and 0.047 in the three evaluation metrics, respectively. As for self-training, the three evaluation metrics show improvements of 0.066, 0.055, and 0.071, respectively. Similar trends can be observed for other ratios of labeled data.

Moreover, the performance of DSSLink not only surpasses the original model for the same dataset proportion but also outperforms the original model for higher dataset proportions. For instance, in Fig. 8, the F1-score of self-training with 10% labeled data (0.595) is higher

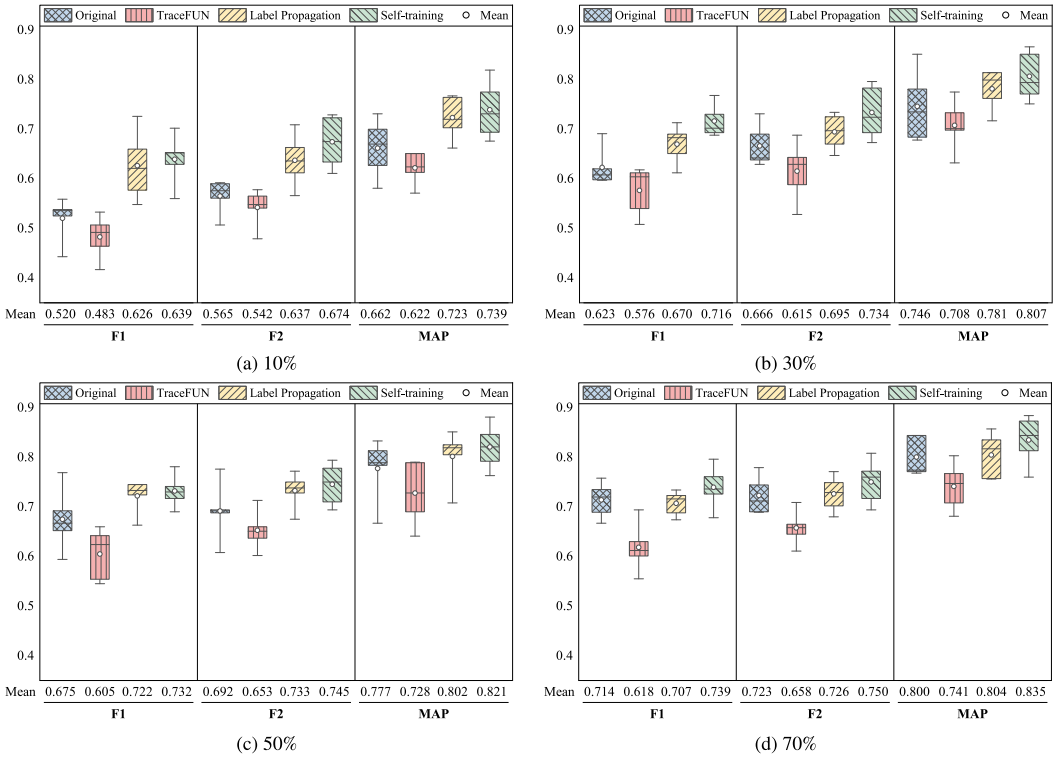

Fig. 9. The evaluation results of Pgcli for original (T-BERT), TraceFUN, and DSSLink (label propagation and self-training).

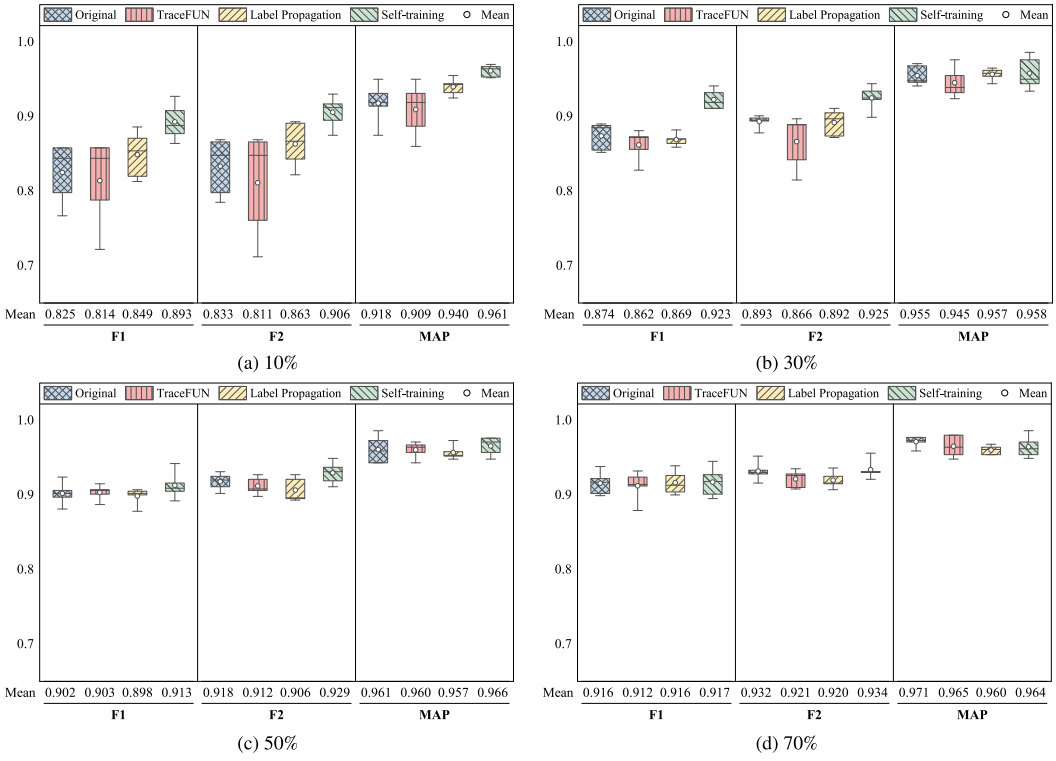

Fig. 10. The evaluation results of Keras for original (T-BERT), TraceFUN, and DSSLink (label propagation and self-training).

than the F1-score of the original model with 30% labeled data (0.572). Similarly, the F1-score of self-training with 30% labeled data (0.616) surpasses the F1-score of the original model with 50% labeled data (0.606), and the F1-score of self-training with 50% labeled data (0.633) outperforms the F1-score of the original model with 70% labeled data (0.612). Similar patterns can be observed in Figs. 9 and 10.

Figs. 11(a), 11(b), and 11(c) show the evaluation results of the Flask, Pgcli, and Keras projects when only 10 labeled data points.

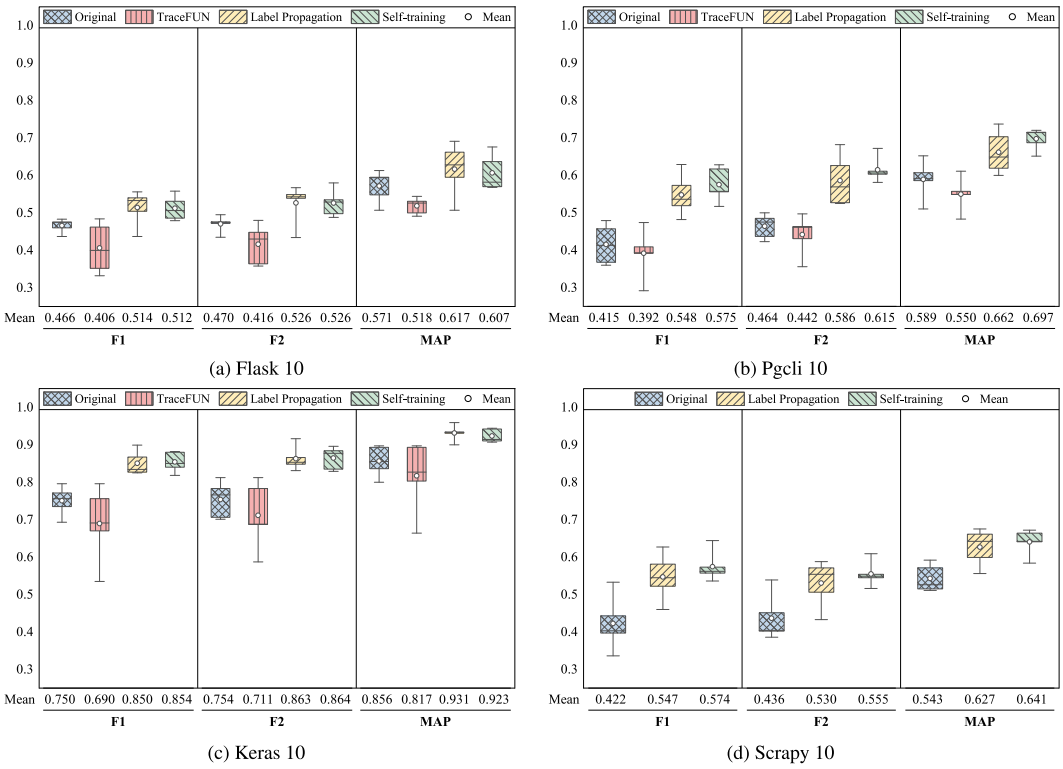

Fig. 11. The evaluation results of Flask, Pgcli, Keras, and Scrapy for original (T-BERT), TraceFUN, and DSSLink (label propagation and self-training).

It can be observed that the performance of DSSLink outperforms the original model. For the Flask project, label propagation achieves an improvement of 0.048 in F1-score, 0.056 in F2-score, and 0.046 in MAP. Similarly, self-training shows improvements of 0.046 in F1-score, 0.056 in F2-score, and 0.036 in MAP. For the Pgcli project, the F1-score, F2-score, and MAP of label propagation increased by 0.133, 0.122, and 0.073, respectively. Self-training achieves even higher improvements of 0.16 in F1-score, 0.151 in F2-score, and 0.108 in MAP. For the Keras project, the F1-score, F2-score, and MAP of label propagation increased by 0.1, 0.109, and 0.075, respectively. The F1-score, F2-score, and MAP of self-training are improved by 0.104, 0.11, and 0.067, respectively.

To validate the generalizability of DSSLink, we assessed the latest SOTA method, BTLink, across 11 Apache projects. By observing the results of T-BERT, we defined a split ratio for the Apache project based on the datasets with better performance improvement (i.e., 10% labeled data). Table 4 shows the evaluation results of the Apache project with a labeled data ratio of 10%. It can be seen that the F1-score and F2-score of DSSLink exceed BTLink in all datasets. Specifically, in the Avro, Beam, log4net, OODT, and Tez datasets, self-training achieved the highest F1-score, which was improved by 0.09, 0.077, 0.068, 0.121, and 0.115, respectively compared to BTLink. In the Buildr, Giraph, Ivy, Isis, Nutch, and Tika datasets, label propagation achieved the highest F1-score, improving by 0.045, 0.106, 0.067, 0.146, 0.025, and 0.116 respectively compared to BTLink.

Tables 5 and 6 illustrate the precision and recall across GitHub and Apache projects with a labeled data ratio of 10%. In GitHub projects, DSSLink outperforms T-BERT in both precision and recall. Specifically, the average precision and average recall of self-training increased by 0.071 and 0.121, respectively; the average precision and average recall of label propagation increased by 0.059 and 0.093, respectively. For Apache projects, DSSLink consistently surpasses BTLink in most cases. However, the performance in terms of recall is slightly poor. Specifically, the average precision and average recall of self-training increased by 0.153 and −0.024, respectively; the average precision and average recall of label propagation increased by 0.127 and 0.012, respectively.

From Figs. 8 to 10, we can find that in the GitHub project, the F1-score of self-training is higher than that of label propagation. In the Flask, Pgcli and Keras datasets, DSSLink uses the self-training to compare with the label propagation, and its average F1-score increases by 0.031, 0.025, and 0.029, respectively. It can be found from Table 4 that in the Apache project, the same phenomenon occurs in the Avro, Beam, log4net, OODT, and Tez datasets, i.e., self-training is better than label propagation. But a different trend appears in the Buildr, Giraph, Ivy, Isis, Nutch, and Tika datasets, where the F1-score of label propagation exceeds that of self-training.

5.2. RQ2: How does DSSLink compare to TraceFUN in improving the performance of TLR when the amount of labeled data is limited?

In our previous work, TraceFUN (Zhu et al., 2022) effectively enhanced the performance of TLR by utilizing unlabeled data in conjunction with the original labeled dataset in Flask, Pgcli, and Keras projects. In this paper, we further explore the impact of dividing the raw labeled data into smaller datasets, representing 10%, 30%, 50%, and 70% of the original labeled data. Figs. 8 to 11 illustrate the performance achieved by TraceFUN on these reduced labeled datasets.

The performance of TraceFUN is inferior to that of DSSLink and even falls below the performance of the original T-BERT model. In the Flask, Pgcli, and Keras projects, the F1-scores obtained by TraceFUN are lower than those of the original model. Specifically, for Flask, the F1-scores across the four datasets (10%, 30%, 50%, and 70%) exhibit decreases of 0.032, 0.029, 0.029, and 0.014, respectively. Similarly, for Pgcli, the F1-scores for the four datasets decrease by 0.037, 0.047, 0.07, and 0.096, respectively. As for Keras, the F1-scores of the three datasets (10%, 30%, and 70%) decrease by 0.011, 0.012, and 0.004, respectively. Moreover, from Figs. 11(a) to 11(c), we can observe that

Table 4
The evaluation results of Apache projects in 10% of labeled datasets.

Dataset	Methods	Metrices		
		F1	F2	MAP
Avro	BTLink	0.744	0.826	0.202
	Self-training	**0.834**	**0.832**	**0.203**
	Label propagation	0.811	0.825	0.201
Beam	BTLink	0.234	0.323	0.067
	Self-training	**0.311**	0.446	0.071
	Label propagation	0.263	**0.523**	**0.079**
Buildr	BTLink	0.795	0.762	0.249
	Self-training	0.832	**0.839**	**0.256**
	Label propagation	**0.840**	0.822	**0.256**
Giraph	BTLink	0.777	0.841	**0.184**
	Self-training	0.861	0.862	**0.184**
	Label propagation	**0.883**	**0.883**	0.182
Ivy	BTLink	0.737	0.781	**0.218**
	Self-training	0.773	**0.786**	**0.218**
	Label propagation	**0.804**	0.763	**0.218**
Isis	BTLink	0.336	0.478	0.192
	Self-training	0.461	0.514	0.201
	Label propagation	**0.482**	**0.535**	**0.204**
log4net	BTLink	0.638	0.741	0.354
	Self-training	**0.706**	0.818	**0.368**
	Label propagation	0.667	**0.861**	0.361
Nutch	BTLink	0.771	0.722	0.191
	Self-training	0.792	**0.792**	0.187
	Label propagation	**0.796**	0.789	**0.192**
OODT	BTLink	0.588	0.691	**0.206**
	Self-training	**0.709**	0.703	**0.206**
	Label propagation	0.649	**0.748**	**0.206**
Tez	BTLink	0.761	0.839	**0.130**
	Self-training	**0.876**	**0.888**	**0.130**
	Label propagation	0.807	0.866	0.128
Tika	BTLink	0.653	0.712	0.207
	Self-training	0.755	0.755	**0.208**
	Label propagation	**0.769**	**0.774**	**0.208**

Table 5
The precision and recall of DSSLink compared to T-BERT on GitHub projects in 10% of labeled datasets.

Dataset	Original		Self-training		Label propagation	
	Precision	Recall	Precision	Recall	Precision	Recall
Flask	0.475	0.585	**0.500**	**0.725**	0.487	0.684
Pgcli	0.537	0.495	**0.618**	0.658	0.585	**0.659**
Keras	0.779	0.867	**0.877**	**0.904**	0.827	0.863
Scrapy	0.467	0.643	0.546	**0.786**	**0.596**	0.755

when there are only 10 labeled data, the F1-scores of the three projects also decrease by 0.06, 0.023 and 0.06, respectively.

In certain datasets, TraceFUN exhibits higher performance than the original model, but it still falls short of the performance achieved by DSSLink. For example, in Fig. 8(c), the maximum value of the F1-score of TraceFUN is higher than that of the original model, but it remains lower than the maximum value obtained by self-training. Similarly, In Fig. 10(c), the average value of the F1-score of TraceFUN (0.903) surpasses that of the original model (0.902) but is still lower than that of self-training (0.913).

5.3. RQ3: What is the impact of the ratio between labeled and unlabeled data on the performance of DSSLink*?*

As mentioned in Section 4.2, due to the limitation in GPU computing power, the maximum batch size was set to 4. Besides, in the training process of deep semi-supervised learning, the amount of unlabeled data should exceed that of labeled data (Sohn et al., 2020; Zhang et al., 2021; Chen et al., 2020). Therefore, each batch consists of one labeled data sample and three unlabeled data samples, resulting in a ratio of three unlabeled samples per one labeled data sample.

Table 6
The precision and recall of DSSLink compared to BTLink on Apache projects in 10% of labeled datasets.

Dataset	Original		Self-training		Label propagation	
	Precision	Recall	Precision	Recall	Precision	Recall
Avro	0.648	**0.874**	0.843	0.825	**0.845**	0.780
Beam	0.163	0.417	**0.216**	0.561	0.155	**0.864**
Buildr	0.865	0.736	0.837	**0.828**	**0.907**	0.782
Giraph	0.702	**0.869**	**0.970**	0.774	0.911	0.857
Ivy	0.700	0.778	0.760	**0.786**	**0.913**	0.718
Isis	0.227	**0.644**	0.457	0.464	**0.520**	0.449
log4net	0.550	**0.759**	**0.818**	0.621	0.720	0.621
Nutch	**0.879**	0.686	0.809	**0.775**	0.832	0.763
OODT	0.475	0.772	**0.790**	0.642	0.548	**0.797**
Tez	0.658	**0.901**	**0.861**	0.891	0.737	0.891
Tika	0.595	0.724	**0.784**	0.729	0.770	**0.768**

Table 7
Comparison of self-training performance utilizing all unlabeled data and partial unlabeled data in 10% labeled dataset.

Dataset	Group	All unlabeled data			Partial unlabeled data		
		F1	F2	MAP	F1	F2	MAP
Flask	1	0.570	0.554	0.672	0.504	0.517	0.650
	2	0.589	0.611	0.690	0.502	0.552	0.650
	3	0.598	0.619	0.669	0.524	0.530	0.595
	4	0.650	0.633	0.737	0.619	0.655	0.701
	5	0.566	0.555	0.653	0.500	0.545	0.606
	avg.	0.595	0.594	0.684	0.530	0.560	0.640
Pgcli	1	0.652	0.723	0.819	0.573	0.610	0.731
	2	0.629	0.675	0.731	0.520	0.575	0.656
	3	0.702	0.729	0.775	0.624	0.642	0.725
	4	0.560	0.611	0.676	0.425	0.503	0.571
	5	0.653	0.634	0.694	0.570	0.617	0.681
	avg.	0.639	0.674	0.739	0.542	0.589	0.673
Keras	1	0.927	0.917	0.970	0.844	0.850	0.941
	2	0.864	0.875	0.967	0.865	0.855	0.946
	3	0.888	0.895	0.953	0.858	0.867	0.928
	4	0.877	0.912	0.952	0.825	0.857	0.923
	5	0.908	0.930	0.964	0.887	0.901	0.950
	avg.	0.893	0.906	0.961	0.856	0.866	0.938

Table 3 illustrates that for the Flask, Pgcli, and Keras projects, when the proportions of labeled data are 50% and 70%, the approximate ratio of unlabeled to labeled software artifacts is 2:1 and 1:1, respectively. This implies that during the training process, there may not be sufficient unlabeled data samples that match the labeled data samples, leading to the repetition of some unlabeled data for training and the potential overfitting of the model. On the other hand, when the proportions of labeled data are 10% and 30%, the quantity of unlabeled data surpasses that of labeled data, satisfying the condition of a three-to-one ratio.

To simulate the lack of unlabeled data, we conducted comparative experiments using the dataset with better TLR performance improvements, i.e., 10% and 30% of labeled data. In these experiments, we adjusted the ratio of labeled and unlabeled software artifacts to 1:1. The experimental results are presented in Tables 7 to 10.

In addition, we collected a new project, i.e., Scrapy, from GitHub and labeled an equal number of trace links as the original labeled dataset of the Flask project. As shown in Table 3, the Scrapy project has a sufficient amount of unlabeled data in datasets with different amounts of labeled data. We evaluated Scrapy to assess the impact of having sufficient unlabeled data on the performance of DSSLink in improving TLR. The experimental results of this evaluation are depicted in Fig. 12.

From Figs. 8 to 10, it can be found that DSSLink may lead to a degradation in TLR performance in certain scenarios where the amount of labeled data is 50% and 70%. Specifically, the F1-score of label

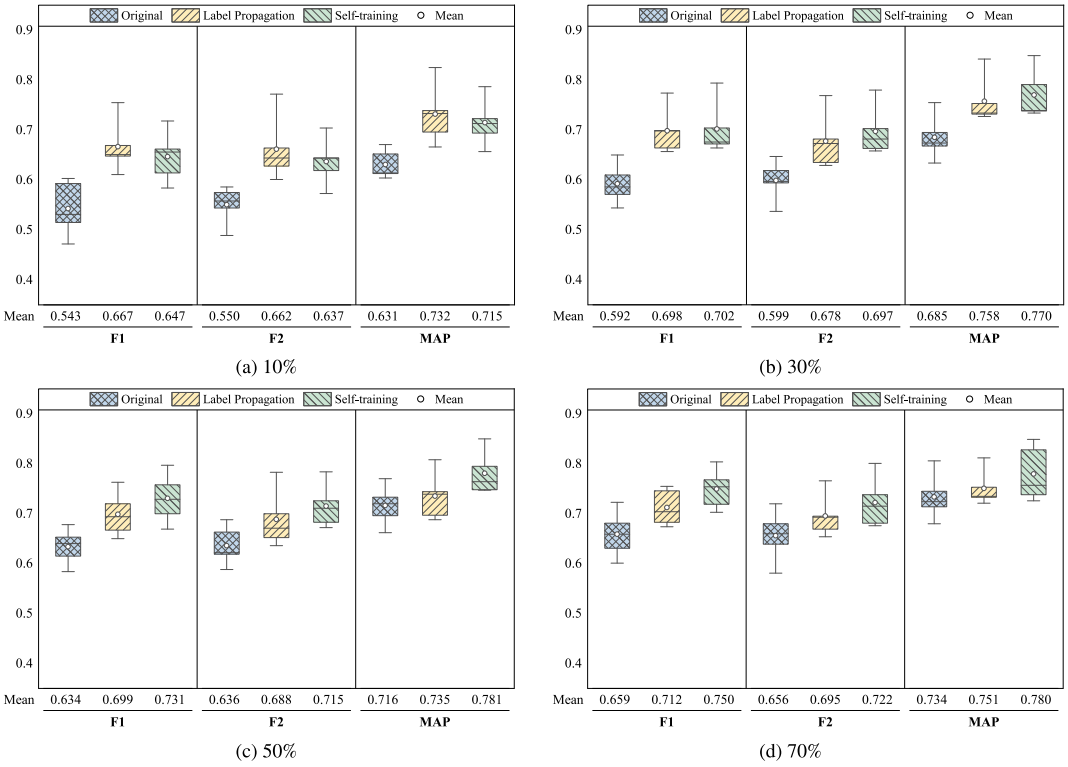

Fig. 12. The evaluation results of Scrapy for original (T-BERT) and DSSLink (label propagation and self-training).

Table 8
Comparison of label propagation performance utilizing all unlabeled data and partial unlabeled data in 10% labeled dataset.

Dataset	Group	All unlabeled data			Partial unlabeled data		
		F1	F2	MAP	F1	F2	MAP
Flask	1	0.565	0.556	0.633	0.529	0.538	0.620
	2	0.538	0.549	0.619	0.534	0.545	0.645
	3	0.576	0.601	0.666	0.547	0.506	0.610
	4	0.629	0.654	0.718	0.592	0.628	0.704
	5	0.556	0.568	0.664	0.506	0.490	0.602
	avg.	0.573	0.586	0.660	0.542	0.541	0.636
Pgcli	1	0.577	0.612	0.720	0.588	0.628	0.706
	2	0.660	0.663	0.764	0.503	0.569	0.643
	3	0.726	0.709	0.767	0.562	0.592	0.659
	4	0.548	0.566	0.662	0.495	0.525	0.581
	5	0.621	0.636	0.703	0.571	0.606	0.670
	avg.	0.626	0.637	0.723	0.544	0.584	0.652
Keras	1	0.820	0.843	0.944	0.817	0.828	0.889
	2	0.813	0.822	0.943	0.824	0.808	0.931
	3	0.871	0.867	0.925	0.882	0.855	0.937
	4	0.854	0.891	0.932	0.807	0.809	0.906
	5	0.886	0.893	0.955	0.852	0.848	0.947
	avg.	0.849	0.863	0.940	0.836	0.830	0.922

Table 9
Comparison of self-training performance utilizing all unlabeled data and partial unlabeled data in 30% labeled dataset.

Dataset	Group	All unlabeled data			Partial unlabeled data		
		F1	F2	MAP	F1	F2	MAP
Flask	1	0.595	0.582	0.708	0.564	0.560	0.671
	2	0.619	0.649	0.724	0.610	0.584	0.659
	3	0.628	0.640	0.716	0.606	0.624	0.703
	4	0.669	0.708	0.756	0.617	0.635	0.723
	5	0.567	0.611	0.697	0.554	0.553	0.636
	avg.	0.616	0.638	0.720	0.590	0.591	0.678
Pgcli	1	0.730	0.796	0.866	0.736	0.784	0.898
	2	0.702	0.724	0.794	0.660	0.691	0.755
	3	0.768	0.783	0.851	0.709	0.748	0.802
	4	0.688	0.673	0.751	0.599	0.656	0.725
	5	0.693	0.693	0.771	0.664	0.718	0.784
	avg.	0.716	0.734	0.807	0.674	0.719	0.793
Keras	1	0.932	0.934	0.976	0.916	0.929	0.970
	2	0.919	0.923	0.950	0.886	0.903	0.941
	3	0.911	0.899	0.934	0.907	0.900	0.957
	4	0.911	0.925	0.944	0.866	0.899	0.924
	5	0.941	0.944	0.986	0.920	0.935	0.967
	avg.	0.923	0.925	0.958	0.899	0.913	0.952

propagation is lower than that of the original model in the Flask, Pgcli, and Keras projects. For Flask, the 50% and 70% labeled datasets decrease by 0.019 and 0.012 in the F1-score, respectively. For Pgcli, the 70% labeled datasets decrease by 0.007 in the F1-score. For Keras, the 50% labeled dataset decrease by 0.004 in the F1-score.

It can be observed from Tables 7 to 10 that reducing the amount of unlabeled data has a significant impact on the performance DSSLink. When using only a portion of the unlabeled data, the performance of DSSLink decreases compared to using all the unlabeled data. In the 10% labeled dataset, self-training exhibits a noticeable decrease in the F1-score, F2-score, and MAP for the Flask, Pgcli, and Keras projects. Specifically, for Flask, the F1-score, F2-score, and MAP decrease by 0.065, 0.034, and 0.044, respectively; for Pgcli, the F1-score, F2-score, and MAP decrease by 0.097, 0.085, and 0.066, respectively; for Keras, the F1-score, F2-score, and MAP decrease by 0.037, 0.04, and 0.023, respectively. Similarly, label propagation experiences a decrease in the evaluation metrics, although the decline is not as pronounced as with self-training. In the 30% labeled dataset, similar patterns are observed, with a decrease in the evaluation metrics for both self-training and label

Table 10
Comparison of label propagation performance utilizing all unlabeled data and partial unlabeled data in 30% labeled dataset.

Dataset	Group	All unlabeled data			Partial unlabeled data		
		F1	F2	MAP	F1	F2	MAP
Flask	1	0.561	0.543	0.656	0.551	0.520	0.607
	2	0.601	0.578	0.643	0.568	0.569	0.655
	3	0.601	0.590	0.649	0.587	0.574	0.659
	4	0.632	0.621	0.718	0.672	0.620	0.696
	5	0.567	0.546	0.608	0.544	0.542	0.639
	avg.	0.592	0.576	0.655	0.584	0.565	0.651
Pgcli	1	0.651	0.697	0.814	0.650	0.701	0.802
	2	0.683	0.725	0.799	0.694	0.708	0.797
	3	0.713	0.734	0.814	0.693	0.733	0.795
	4	0.612	0.647	0.717	0.571	0.621	0.690
	5	0.690	0.670	0.762	0.688	0.693	0.775
	avg.	0.670	0.695	0.781	0.659	0.691	0.772
Keras	1	0.882	0.911	0.965	0.919	0.941	0.973
	2	0.864	0.874	0.962	0.900	0.902	0.937
	3	0.870	0.872	0.954	0.912	0.902	0.936
	4	0.859	0.905	0.944	0.863	0.883	0.915
	5	0.870	0.897	0.958	0.889	0.911	0.964
	avg.	0.869	0.892	0.957	0.897	0.908	0.945

propagation. However, in the Keras project, label propagation does not exhibit a decline in the three evaluation metrics.

It can be observed from Figs. 11(d) and 12 that the TLR performance of the Scrapy project is improved using DSSLink. Even when the amount of labeled data is increased to 50% and 70%, the performance of DSSLink remains higher than that of the original model. In all four datasets of Scrapy (10%, 30%, 50%, and 70%), both label propagation and self-training achieve higher F1-scores, F2-scores, and MAPs compared to the original model. For example, for all four datasets of Scrapy, the F1-score of label propagation increases by 0.124, 0.106, 0.065, and 0.053 for the respective datasets. Similarly, the F1-score of self-training increased by 0.104, 0.11, 0.097, and 0.091. Even in scenarios with only 10 labeled data, both label propagation and self-training demonstrate improvements with the F1-score increasing by 0.125 and 0.152, respectively.

5.4. RQ4: What is the impact of the pairing rule of unlabeled data on the performance of DSSLink?

The impact of different pairing rules of unlabeled software artifacts on the performance of DSSLink was evaluated by randomly selecting the same number of artifact pairs from the pairing results of the Cartesian product as the unlabeled data selected by the time-based rule (described in Section 3.3). The comparative experiments were conducted to assess the performance differences resulting from the two different pairing rules.

From Tables 11 and 12, it can be found that the performance of DSSLink using the time-based rule is significantly better than that of the random-based rule. In both 10% and 30% labeled datasets, the F1-score of the random-based rule is significantly lower than that of the time-based rule for both self-training and label propagation. The F1-scores of the Flask, Pgcli, and Keras projects show a notable drop when using the random-based rule. For example, for self-training, the F1-score decreases by 0.092, 0.109, and 0.095 for the Flask, Pgcli, and Keras projects, respectively, in the 10% labeled datasets. For label propagation, the F1-score decreases by 0.116, 0.119, and 0.068 for the Flask, Pgcli, and Keras projects, respectively, in the 10% labeled datasets. Moreover, in both cases, the scores obtained using the random-based rule are even worse than those of the original model.

5.5. RQ5. How significantly does DSSLink impact the computational efficiency of traceability methods?

Tables 13 and 14 present the computation time of DSSLink (i.e., self-training and label propagation), along with the computation time of the original traceability methods (i.e., T-BERT and BTLink). The computation time of the original method encompasses the training and evaluation phases. In contrast, the computation time of the DSSLink model includes the pre-training and the training of deep semi-supervised learning stages, along with their respective evaluation stages. It is worth noting that as mentioned in Section 4.2, the epochs for both pre-training and deep semi-supervised are only half of the original model, and their sum is equal to the epochs of the original model.

We used 15 projects for evaluation, and the proportion of labeled data is 10%. In this evaluation, T-BERT is assessed using four GitHub projects on a machine equipped with GPU Nvidia RTX 2080Ti. BTLink is evaluated using 11 Apache projects. Because this method requires high computing resources, we used the same GPU configuration as in the work of Lan et al. (2023). The original BTLink model and the self-training model of DSSLink were conducted on a machine equipped with Nvidia RTX 3090. In addition, label propagation requires more graphic memory for graph operations. Therefore, we conducted the experiment of label propagation model of DSSLink on a machine equipped with Nvidia A100-PCIE-40 GB.

It can be observed from Tables 13 and 14 that, for the Flask, Pgcli, Keras, and Scrapy datasets, the computation time of DSSLink's self-training model is 6.0X, 3.9X, 6.3X, and 12.6X that of the T-BERT model, respectively. The computation time of DSSLink's label propagation model is 5.8X, 3.9X, 6.1X, and 12.0X that of the T-BERT model, respectively. The time difference between self-training and label propagation for enhancing the T-BERT model is not substantial.

In addition, in the Avro, Beam, Buildr, Giraph, Ivy, Isis, log4net, Nutch, OODT, Tez, and Tika datasets, the computation time of DSSLink's self-training model is 1.8X, 1.4X, 1.7X, 1.8X, 1.8X, 1.8X, 1.9X, 1.8X, 1.8X, 1.8X, and 1.9X that of the BTLink model, respectively. The computation time of DSSLink's label propagation model is 1.2X, 1.1X, 1.3X, 1.5X, 1.4X, 1.4X, 1.5X, 1.4X, 1.3X, 1.3X, and 1.3X that of the BTLink model, respectively. Compared to the T-BERT model, the time difference between self-training and label propagation is large.

6. Discussions

This section analyzes the experimental results for the five RQs and discusses the significance of our findings.

6.1. RQ1: Can DSSLink enhance the performance of TLR when the amount of labeled data is limited?

The experimental results from the Flask, Pgcli, and Keras projects demonstrate that the DSSLink improves TLR performance significantly, especially when the amount of labeled data is limited. With varying proportions of labeled data volumes (10%, 30%, 50%, and 70%), DSSLink consistently outperforms using labeled data alone, even when only a small number (i.e., 10) of labeled links are available. This is particularly beneficial in real-world TLR tasks where labeled datasets are often limited, as DSSLink leverages deep semi-supervised learning to effectively utilize both labeled and unlabeled data, resulting in improved TLR performance beyond what can be achieved using labeled data alone.

Moreover, we observed that DSSLink's performance in datasets with a low proportion of labeled data can surpass the performance of the original model in datasets with a high proportion of labeled data. For example, the performance of self-training with 10% labeled data outperforms the performance of the original model with 30% labeled data. This finding highlights the significant impact of DSSLink in enhancing

Table 11
The average of F1-score, F2-score and MAP based on different unlabeled artifacts pairing rules in 10% labeled dataset.

Dataset	Original			Time-based rule						Random-based rule					
				Self-training			Label propagation			Self-training			Label propagation		
	F1	F2	Map	F1	F2	Map	F1	F2	Map	F1	F2	Map	F1	F2	Map
Flask	0.529	0.539	0.613	0.595	0.594	0.684	0.573	0.586	0.660	0.503	0.507	0.580	0.457	0.453	0.517
Pgcli	0.520	0.565	0.662	0.639	0.674	0.739	0.626	0.637	0.723	0.530	0.552	0.631	0.507	0.524	0.596
Keras	0.825	0.833	0.918	0.893	0.906	0.961	0.849	0.863	0.940	0.798	0.795	0.890	0.781	0.785	0.866

Table 12
The average of F1-score, F2-score and MAP based on different unlabeled artifacts pairing rules in 30% labeled dataset.

Dataset	Original			Time-based rule						Random-based rule					
				Self-training			Label propagation			Self-training			Label propagation		
	F1	F2	Map	F1	F2	Map	F1	F2	Map	F1	F2	Map	F1	F2	Map
Flask	0.572	0.571	0.656	0.616	0.638	0.720	0.592	0.576	0.655	0.533	0.535	0.617	0.524	0.521	0.595
Pgcli	0.623	0.666	0.746	0.716	0.734	0.807	0.670	0.695	0.781	0.614	0.649	0.739	0.590	0.601	0.663
Keras	0.874	0.893	0.955	0.923	0.925	0.958	0.869	0.892	0.957	0.869	0.873	0.938	0.841	0.841	0.917

Table 13
The computation time of DSSLink compared to T-BERT on GitHub projects in 10% of labeled datasets.

Dataset	Computation time (min)		
	Original	Self-training	Label propagation
Flask	36.6	220.6	212.3
Pgcli	24.1	93.4	92.9
Keras	35.3	223.4	214.5
Scrapy	46.3	583.4	554.2

Table 14
The computation time of DSSLink compared to BTLink on Apache projects in 10% of labeled datasets.

Dataset	Computation time (min)		
	Original	Self-training	Label propagation
Avro	49.1	89.1	57.3
Beam	506.8	713.7	561.6
Buildr	18.8	32.2	24.7
Giraph	34.5	61.0	51.6
Ivy	24.4	43.2	32.9
Isis	154.4	273.5	211.9
log4net	4.1	7.6	6.3
Nutch	32.9	60.7	44.3
OODT	21.9	38.9	28.6
Tez	73.1	129.9	94.6
Tika	53.3	99.2	71.0

TLR performance. The deep neural networks trained by DSSLink, utilizing only 10% labeled data alongside all unlabeled data, outperform supervised TLR models trained using 30% labeled data alone. DSSLink has the ability to extract more information from a large amount of unlabeled data compared to supervised TLR methods that rely on partially labeled data.

In 11 Apache projects, DSSLink effectively enhances the BTLink method, confirming its robust generalization. Despite using distinct BERT models and model structures from T-BERT and BTLink, DSSLink demonstrates performance improvements for both. DSSLink attains favorable outcomes across four GitHub projects and 11 Apache projects. Specifically, in the datasets with a label ratio of 10%, the average improvement of F1-score is 15.7% in the GitHub projects, with a maximum improvement of 22.9%. For the Apache projects, the average improvement of F1-score is 15.2%, with a maximum improvement of 43.5%. Experimental results unequivocally establish the effectiveness of DSSLink. This approach leverages deep semi-supervised learning to extract knowledge from unlabeled data, showcasing its effectiveness across the two SOTA traceability methods and diverse datasets.

The precision and recall of DSSLink significantly surpassed those of T-BERT in the GitHub projects. While in the Apache projects, there were cases where it performed slightly less effectively than BTLink. However, regarding precision and recall metrics, there was no scenario where both indicators were lower than BTLink. Specifically, for the GitHub projects, the average improvements of precision and recall are 12.5% and 18.6%, respectively. For the Apache projects, the average improvements of precision and recall are 26.1% and 1.6%, respectively. The comparison results of precision and recall further demonstrate the effectiveness of DSSLink in enhancing traceability methods.

Moreover, the results indicate that there is no clear superiority or inferiority between the deep semi-supervised learning algorithms of self-training and label propagation. When comparing their enhancements to traceability methods, the performance differences between the two algorithms are insignificant. Self-training primarily focuses on unknown data, learning general rules from labeled data in the current dataset to predict new data outside the current dataset. On the other hand, label propagation emphasizes existing data, propagating label information from the current dataset to unlabeled nodes to predict unlabeled data within the current dataset. Users can choose different deep semi-supervised learning methods in DSSLink based on their specific training goals.

6.2. RQ2: How does DSSLink *compare to TraceFUN in improving the performance of TLR when the amount of labeled data is limited?*

When there is a similar relationship between a labeled artifact and an unlabeled artifact of the same type in a trace link, there is a high probability that the unlabeled artifact is also connected to another artifact in the link. TraceFUN (Zhu et al., 2022) leverages this similarity between artifacts to create new links. For example, if there is a trace link between an issue and a commit, this method identifies similar relationships between the commit and other unlabeled commits. It selects the most similar relationship and establishes a new link between the unlabeled commit and the labeled issue. When more labeled data is available, there are more retrievable similar relationships, resulting in a large number of high-quality newly created links. By incorporating this expanded training set, TraceFUN effectively enhances the TLR performance.

In previous work (Zhu et al., 2022), TraceFUN's effectiveness was attributed to the utilization of a large number of labeled links. The number of links in the Flask, Pgcli, and Keras projects reached 481, 338, and 353, respectively. These links facilitated the generation of numerous artifact pairs, increasing the likelihood of retrieving pairs with similar relationships. However, as the number of artifact pairs decreases, so does the number of high-similarity pairs. For example,

with 10 high-similarity pairs among 100, reducing the total pairs to 50 implies having 10 or fewer high-similarity pairs. Consequently, when relying on a limited set of labeled links, TraceFUN may struggle to obtain enough similar relationships to create new links, leading to diminished performance.

By contrast, in the context of this study, when the labeled data constitutes 70% of the dataset, the link counts for projects like Flask, Pgcli, and Keras are 336, 236, and 247, respectively. Other datasets with different proportions have even fewer links, e.g., only 10 labeled links. This scenario results in the generation of a small number of artifact pairs. Subsequently, a reduced number of artifact pairs with similar relationships. Consequently, the new links created by TraceFUN either consist of a limited number of true links, leading to minimal changes in model performance, or introduce a significant number of false positives, resulting in a degradation in model performance.

TraceFUN demonstrates effectiveness in enhancing TLR performance when there is an ample amount of labeled data, as indicated in Zhu et al. (2022). However, when confronted with a scarcity of labeled data, TraceFUN encounters challenges and may even lead to a deterioration in TLR performance. The comparison between DSSLink and TraceFUN shows that DSSLink effectively overcomes the limitation of TraceFUN.

6.3. RQ3: What is the impact of the ratio between labeled and unlabeled data on the performance of DSSLink?

In our experiments, we divided the original labeled dataset into subsets with different proportions of labeled and unlabeled data. RQ3 aimed to analyze how these varying ratios impact the performance of DSSLink in improving TLR. As shown in Table 3, for the Flask, Pgcli, and Keras projects, in datasets with 10% and 30% labeled data, the amount of unlabeled data is larger, at least 3X that of labeled data. At this time, DSSLink can effectively improve TLR performance. However, in datasets with 50% and 70% labeled data, the approximate proportions of unlabeled and labeled data are 2:1 and 1:1, respectively. The difference in the quantity of labeled and unlabeled data is not significant. DSSLink can only marginally improve TLR performance or may even result in a degradation of TLR performance.

By contrast, As depicted in Table 3, the newly collected Scrapy dataset contains a substantial amount of unlabeled data. In the datasets with labeled data proportions of 50% and 70%, the quantity of unlabeled data is roughly 5X and 3X that of labeled data, respectively. We maintained the number of labeled data points consistent with the Flask project to simulate a scenario of abundant unlabeled data. The results demonstrated that DSSLink consistently enhances TLR performance across all proportions of the Scrapy project's datasets, as long as the amount of unlabeled data far exceeds the labeled data.

To further validate our observations, we reduced the quantity of unlabeled data in the Flask, Pgcli, and Keras datasets (i.e., 10% and 30% labeled data) where TLR performance improvement was more pronounced. This simulates a scenario of insufficient unlabeled data. We can see from Tables 7 to 10 that the outcomes revealed a significant decrease in DSSLink's performance when utilizing partial unlabeled data compared to all unlabeled data.

The effectiveness of DSSLink is contingent on the quantity of unlabeled data. DSSLink demonstrates its capacity to significantly enhance TLR performance only when the amount of unlabeled data substantially exceeds that of labeled data. This necessity arises due to the intrinsic requirements of deep semi-supervised learning, which necessitates learning from a limited set of labeled data and a considerable amount of unlabeled data. Sufficiently large volumes of unlabeled data are crucial for acquiring meaningful and effective information. When the amount of unlabeled data is limited, such as when a substantial portion is filtered out through a confidence threshold, it can be detrimental to model learning. Leveraging more unlabeled data whenever possible contributes to overall performance improvement (Chen et al., 2023).

In summary, the key factor that changes the performance of DSSLink between 10% and 30% labeled datasets and performs better in Scrapy projects is the amount of unlabeled data. In real-world projects, labeling data is often limited due to cost considerations, while a significant amount of unlabeled data is available. In such cases, applying DSSLink can lead to significant improvements in TLR performance compared to relying solely on limited labeled data.

6.4. RQ4: What is the impact of the pairing rule of unlabeled data on the performance of DSSLink?

The performance of the time-based rule in pairing unlabeled software artifacts is superior to that of the random-based rule, indicating the advantage of using temporal rules. In the selection process, i.e., pseudo-data selection in DSSLink, unlabeled data that are considered positive samples through the inference of deep semi-supervised learning is crucial because it contains both true positive and false positive samples, with true positive samples being beneficial for network learning, while false positive samples can mislead the learning process.

In the Cartesian product pairing approach for unlabeled software artifacts, there is a severe class imbalance issue with a disproportionate number of data samples having true label 1 compared to true label 0. While randomly selecting some software artifact pairs addresses efficiency concerns, it exacerbates the class imbalance problem. By selecting only a portion of the data, the number of instances with a true label of 1 becomes very limited, which hampers the network's learning process. Therefore, the random-based rule can lead to a decline in the TLR performance of DSSLink.

The time-based rule in DSSLink prioritizes retaining software artifact pairs with a real label of 1 (i.e., true positive samples) while reducing the inclusion of pairs with a real label of 0 (i.e., true negative samples). This approach ensures a larger number of true positives, which enhances network learning. By minimizing the number of software artifact pairs with a true label of 0, the occurrence of false positives is reduced, minimizing their negative impact on network learning. Therefore, employing the time-based rule in DSSLink effectively enhances TLR performance.

6.5. RQ5. How significantly does DSSLink impact the computational efficiency of traceability methods?

The experimental results reveal a significant increase in computation time when DSSLink enhances T-BERT. While the computation time for the BTLink model is not substantially increased, it remains higher than that of the original model. This outcome is predictable. The computation time for the original traceability method is confined to training on a limited set of labeled data. With DSSLink, the computational time encompasses not only training on labeled data but also the inference of pseudo-labels for unlabeled data in each iteration of training, followed by training on these pseudo-labeled instances. Consequently, DSSLink extends the computation time compared to the original traceability method, attributed to the additional steps of deep semi-supervised learning and training on pseudo-labeled data.

As indicated by the time results of T-BERT, there is no significant difference in the computation time between DSSLink's self-training and label propagation models, since their experiments were conducted on the same GPU. However, when examining the time results of BTLink, it becomes apparent that DSSLink's label propagation model requires less computation time than the self-training model. This is because the GPUs used in their experiments are different. As mentioned in Section 4.2, due to the computing resource requirements, the label propagation model for BTLink uses a more powerful GPU to run compared to the self-training model.

Moreover, the computation time increase of T-BERT is significantly larger than that of BTLink. This is because their training parameters are different. As mentioned in Section 4.2, the epochs of T-BERT's original model are 200, and DSSLink model for it includes 100 epochs of pre-training and 100 epochs of deep semi-supervised learning. The epochs of BTLink are only 1/10 of T-BERT.

In summary, DSSLink increases the computation time of traceability methods. Despite the reduced computational efficiency compared to the original method, DSSLink enhances TLR performance. When compared to enhancing performance through manual annotation of traceability links, our method eliminates the need for labor costs, making it a more efficient approach. Therefore, given the performance improvement achieved by DSSLink, the reduction in computational efficiency is acceptable. It is important to note that the extent of the computation time added by DSSLink varies significantly for different traceability methods. For instance, in the case of BTLink, our method increases the computation time by less than double while delivering improvements in TLR performance.

7. Threats to validity

Threats to Internal Validity. The threat to the internal validity mainly comes from the quality of pseudo-labeled data. To address the potential threat, we follow the same strategy as the work of Lee et al. (2013) and Iscen et al. (2019). In self-training, the confidence threshold is used to filter unconfident predictions of the model, and only samples with prediction probabilities higher than the threshold participate in the loss calculation. In label propagation, each pseudo-labeled data is weighted to diminish the influence of samples with higher uncertainty in the loss calculation. Although these strategies mitigate the adverse impact of erroneous pseudo-label data on the model, they do not entirely eliminate the introduction of false pseudo-labels. Therefore, more precise pseudo-label quality rules still need to be explored in future research.

Besides, the initial labeled data can also threaten internal validity. Deep semi-supervised learning relies on acquiring knowledge from a limited labeled dataset and generalizing it to unlabeled data. If the initial labeled dataset lacks representative samples or contains biased data, the pre-trained model may perform poorly, rendering it ineffective in deep semi-supervised learning. To mitigate this threat, we choose the datasets provided by Lin et al. (2021) and Lan et al. (2023) for experiments. These datasets have been experimentally verified to be valid in their studies.

Moreover, internal validity threat may also arise from re-opening issues. For example, if certain issues in the existing dataset are re-opened, their closing time may change. Upon recollection, the creation and closing time ranges of these issues will be extended compared to the previous collection. The extension in time ranges could result in more commits falling within them. Consequently, DSSLink may generate additional pairs when pairing unlabeled artifacts. This change potentially impacts the performance of DSSLink.

Threats to External Validity. The threat to the external validity mainly lies in the generalizability of DSSLink. To address this threat, we have selected T-BERT (Lin et al., 2021) and BTLink (Lan et al., 2023), two SOTA deep learning-based TLR methods, as our baseline for comparison. Besides, the effectiveness of DSSLink is assessed on 15 projects, including four GitHub projects and 11 Apache projects. We ensure a fair and direct comparison of DSSLink's performance.

Threats to Construct Validity. The choice of evaluation metrics for predictive performance can pose a threat to construct validity. To reduce this threat, we have used precision, recall, F-scores and MAP.

8. Related work

8.1. TLR for issues and commits

The recovery of missing links between issues and commits has been the focus of research, leading to the proposal of various methods. Initially, the traditional approach relied on identifiers in issues and commits, but it was found to lose many links due to developers' oversight of adding identifiers (Bachmann et al., 2010). Wu et al. (2011) introduced ReLink, an automated method that retrieves similar text between issues and commits based on the explicit link. However, the textual differences between issues and commits pose a challenge. To address this, Nguyen et al. (2012) proposed MLink, which extracts text from source code, including code identifiers and comments, to enhance the text similarity with issues. Researchers have also employed machine learning techniques for link recovery. Le et al. (2015) developed RCLinker, which generates commits with rich context using ChangeScribe and predicts the link probability using a classification model. Sun et al. (2017b) presented FRLink, a method based on file correlation that identifies the relevant files for fixing the error.

While the above methods rely on textual similarity, recent studies have explored the application of deep learning models to enhance TLR performance. Ruan et al. (2019) proposed DeepLink, which employs word embedding and recurrent neural networks (RNN) to capture the semantic relationship between issues and commits. Lin et al. (2021) introduced T-BERT, utilizing a BERT-based relational classifier to restore links by leveraging pre-trained language models and transfer learning, achieving superior performance compared to RNN-based approaches. Lan et al. (2023) used pre-trained language models to automatically recover links. They used two different BERT models to represent the feature vector of software artifacts, the RoBERTa model for receiving natural language and the CodeBERT model for receiving programming language.

8.2. Unlabeled data for TLR

The effectiveness of deep learning models heavily rely on the availability of large labeled datasets, which can be costly to obtain in real-world projects. As a result, researchers have started exploring the potential of leveraging abundant unlabeled data to enhance TLR performance.

Sun et al. (2017a) proposed PULink. In PULink, true links are employed as positive instances, and unlabeled links are regarded as negative instances for training a traditional classifier. Subsequently, the traditional classifier is transformed into a binary classifier based on the data distribution. This binary classifier could then be applied to determine the labels of links in unlabeled data. This means that PULink utilizes unlabeled data directly as training data. By contrast, unlabeled data in DSSLink is not directly used to train the model. Pseudo-labels of unlabeled data should be obtained through deep semi-supervised learning before being utilized for model training. By contrast, unlabeled data in DSSLink is not directly used to train the model. Pseudo-labels of unlabeled data should be obtained through deep semi-supervised learning before being utilized for model training.

Mills et al. (2019) integrated active learning into the TLR domain. They initiated training with limited labeled data, leveraging the model to predict unlabeled data and calculate information entropy based on predicted probabilities. Data with high information entropy is subsequently presented to human experts for labeling. This approach enables the model to actively choose uncertain data for learning, effectively reducing labor costs. However, their semi-automated method still involves manual annotation, whereas our approach is fully automated.

Dong et al. (2022) addressed the challenges of class imbalance and data sparsity by employing adjusted class-balancing and self-training policies, thereby enhancing traceability methods based on machine

learning. However, their method needs to extract 27 data features for machine learning model training. These features include textual similarity features, standalone textual similarity, and hybrid textual similarity. In contrast, our research focuses on enhancing traceability methods based on deep learning. We use deep features automatically extracted by deep neural networks for pseudo-label inference.

The abovementioned existing studies focus on using unlabeled data to enhance machine learning-based TLR methods. In our previous work, we proposed TraceFUN (Zhu et al., 2022), concentrating on enhancing deep learning-based TLR methods. TraceFUN is introduced to label unlabeled data before TLR model training, enhancing the training data in a one-time process. As mentioned in the third paragraph of the introduction, the core idea of TraceFUN is that when an unlabeled artifact shares a similar relationship with a labeled artifact, it is highly probable that it is also connected to the linked objects associated with the labeled artifact. This implies that TraceFUN heavily relies on the similar relationship between artifacts to generate new links. When the amount of labeled data is small, retrieving artifact pairs with high similarity is difficult, resulting in poor-quality link generation. Our experiment results further prove that when reducing the amount of labeled data, TraceFUN will suffer significant performance degradation. By contrast, DSSLink continuously infers pseudo-labels of unlabeled data throughout the deep semi-supervised learning process, constituting an iterative approach. Our experiment results show that DSSLink overcomes the limitation of TraceFUN.

8.3. Deep semi-supervised learning

Semi-supervised learning (SSL), which leverages both labeled and unlabeled data, has emerged as a powerful approach for achieving high performance with limited samples. In recent years, there has been growing interest in integrating SSL into the training of deep neural networks to address the reliance of high-performance networks on large labeled datasets. Two prominent techniques in deep semi-supervised learning are entropy minimization and consistency regularization. Entropy minimization focuses on ensuring that the decision boundary passes through areas where data is sparsely distributed, leading to low entropy predictions that are close to a specific category. Lee et al. (2013) introduced the pseudo-label method, where they utilized the current model to predict pseudo-labels for unlabeled samples based on their maximum probability, and employed entropy minimization to approximate one of the class probabilities predicted by the pseudo-labeled model, thus minimizing the conditional entropy of the pseudo-labels. Consistency regularization, on the other hand, aims to ensure that the model produces consistent output distributions when subjected to perturbations in input. Xie et al. (2020) proposed unsupervised data enhancement, where they introduced noise to enhance unlabeled data and applied consistency regularization to minimize the loss between the original unlabeled data and its enhanced version. Another method, Mixmatch, proposed by Berthelot et al. (2019b), also utilizes consistency regularization. However, they employed a novel data augmentation technique that combines the hidden vectors of the data to generate new data instances with labels that are a combination of the original data labels.

Moreover, deep semi-supervised learning can leverage deep features obtained from deep neural networks to facilitate SSL algorithms. Iscen et al. (2019) combined label propagation with deep learning, where they inferred pseudo-labels for unlabeled data through label propagation during the training of deep neural networks. These pseudo-labeled data were then combined with the labeled data to retrain the network. By iterating this process, they achieved a high-performance network.

Deep semi-supervised learning techniques have found applications in the field of software engineering as well. Yu et al. (2021) introduced a text classification method based on semi-supervised transfer learning. Their approach involves data augmentation on unlabeled data, encouraging the model to make high-confidence predictions on both the original unlabeled data and its augmented counterparts to minimize entropy. This method achieves impressive performance even with a small number of labeled samples. Dong et al. (2023) proposed the MixCode method, drawing inspiration from Mixup techniques in computer vision. They first reconstruct the original code and then generate new training data by performing Mixup between the original and reconstructed code. This technique effectively expands the training dataset and enables consistent training of deep neural networks, leading to enhanced performance in code analysis tasks.

To the best of our knowledge, this paper is the first to apply deep semi-supervised learning to the TLR domain.

9. Conclusions

This paper presents DSSLink, a novel approach for enhancing TLR performance using deep semi-supervised learning techniques. It is specifically crafted for tasks characterized by a scarcity of labeled data and an abundance of unlabeled data. It learns knowledge from labeled data through a pre-trained model, leverages knowledge from unlabeled data via deep semi-supervised learning, and ultimately combines the two by a joint loss mechanism to recover traceability links. DSSLink effectively harnesses the rich latent information present in unlabeled data through deep semi-supervised learning, resulting in an enhanced TLR performance. Our evaluation across 15 open-source software projects reveals that DSSLink outperforms T-BERT and BTLink in TLR performance. Notably, DSSLink successfully addresses the limitation of TraceFUN under a scenario of limited labeled data. The source code and dataset of DSSLink have been released at https://github.com/DSSLink.

CRediT authorship contribution statement

Jianfei Zhu: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing, Funding acquisition. **Guanping Xiao:** Conceptualization, Data curation, Funding acquisition, Investigation, Methodology, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Zheng Zheng:** Funding acquisition, Validation, Writing – review & editing. **Yulei Sui:** Funding acquisition, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

https://github.com/DSSLink.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by the National Natural Science Foundation of China under Grants 62002163 and 61772055, Natural Science Foundation of Jiangsu Province under Grant BK20200441, Australian Research Council under Grants DP210101348 and FT220100391, and Postgraduate Research & Practice Innovation Program of NUAA under Grant XCXJH20221615.

References

Arnold, A., Nallapati, R., Cohen, W.W., 2007. A comparative study of methods for transductive transfer learning. In: Proceedings of the 7th IEEE International Conference on Data Mining Workshops. ICDMW, IEEE, pp. 77–82.

Bachmann, A., Bird, C., Rahman, F., Devanbu, P., Bernstein, A., 2010. The missing links: bugs and bug-fix commits. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. FSE, pp. 97–106.

Behnamghader, P., Alfayez, R., Srisopha, K., Boehm, B., 2017. Towards better understanding of software quality evolution through commit-impact analysis. In: Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security. QRS, IEEE, pp. 251–262.

Belkin, M., Niyogi, P., Sindhwani, V., 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. J. Mach. Learn. Res. 7 (11).

Berthelot, D., Carlini, N., Cubuk, E.D., Kurakin, A., Sohn, K., Zhang, H., Raffel, C., 2019a. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. In: International Conference on Learning Representations. ICLR.

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., Raffel, C.A., 2019b. Mixmatch: A holistic approach to semi-supervised learning. Adv. Neural Inf. Process. Syst. 32.

Bugzilla, 2023. A web-based general-purpose bug tracking system and testing tool. https://www.bugzilla.org/.

Chapelle, O., Schölkopf, B., Zien, A. (Eds.), 2006. Semi-Supervised Learning. The MIT Press, ISBN: 9780262033589, URL http://dblp.uni-trier.de/db/books/collections/CSZ2006.html.

Chapelle, O., Weston, J., Schölkopf, B., 2002. Cluster kernels for semi-supervised learning. Adv. Neural Inf. Process. Syst. 15.

Chen, Y., Tan, X., Zhao, B., Chen, Z., Song, R., Liang, J., Lu, X., 2023. Boosting semi-supervised learning by exploiting all unlabeled data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, pp. 7548–7557.

Chen, J., Yang, Z., Yang, D., 2020. MixText: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, pp. 2147–2157.

Dong, Z., Hu, Q., Guo, Y., Cordy, M., Papadakis, M., Zhang, Z., Le Traon, Y., Zhao, J., 2023. MixCode: Enhancing code classification by mixup-based data augmentation. In: Proceedings of the 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 379–390.

Dong, L., Zhang, H., Liu, W., Weng, Z., Kuang, H., 2022. Semi-supervised pre-processing for learning-based traceability framework on real-world software projects. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE, pp. 570–582.

Git, 2023. A free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. https://git-scm.com/.

Hayes, B.K., Heit, E., Swendsen, H., 2010. Inductive reasoning. Wiley Interdiscip. Rev.: Cogn. Sci. 1 (2), 278–292.

Heinemann, L., Hummel, B., Steidl, D., 2014. Teamscale: Software quality control in real-time. In: Proceedings of the 36th International Conference on Software Engineering. ICSE, pp. 592–595.

Iscen, A., Tolias, G., Avrithis, Y., Chum, O., 2019. Label propagation for deep semi-supervised learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, pp. 5070–5079.

Lan, J., Gong, L., Zhang, J., Zhang, H., 2023. Btlink: automatic link recovery between issues and commits based on pre-trained BERT model. Empir. Softw. Eng. 28 (4), 103.

Le, T.-D.B., Linares-Vásquez, M., Lo, D., Poshyvanyk, D., 2015. Rclinker: Automated linking of issue reports and commits leveraging rich contextual information. In: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension. ICPC, IEEE, pp. 36–47.

Lee, D.-H., et al., 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In: Workshop on Challenges in Representation Learning. ICML, Vol. 3, p. 896.

Li, J., He, P., Zhu, J., Lyu, M.R., 2017. Software defect prediction via convolutional neural network. In: Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security. QRS, IEEE, pp. 318–328.

Liao, Z., He, D., Chen, Z., Fan, X., Zhang, Y., Liu, S., 2018. Exploring the characteristics of issue-related behaviors in github using visualization techniques. IEEE Access 6, 24003–24015.

Lin, J., Liu, Y., Zeng, Q., Jiang, M., Cleland-Huang, J., 2021. Traceability transformed: Generating more accurate links with pre-trained bert models. In: Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 324–335.

Mazrae, P.R., Izadi, M., Heydarnoori, A., 2021. Automated recovery of issue-commit links leveraging both textual and non-textual data. In: Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 263–273.

Mills, C., Escobar-Avila, J., Bhattacharya, A., Kondyukov, G., Chakraborty, S., Haiduc, S., 2019. Tracing with less data: active learning for classification-based traceability link recovery. In: Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 103–113.

Miyato, T., Maeda, S.-i., Koyama, M., Ishii, S., 2018. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. IEEE Trans. Pattern Anal. Mach. Intell. 41 (8), 1979–1993.

Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N., 2012. Multi-layered approach for recovering links between bug reports and fixes. In: Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. FSE, pp. 1–11.

Ruan, H., Chen, B., Peng, X., Zhao, W., 2019. DeepLink: Recovering issue-commit links based on deep learning. J. Syst. Softw. 158, 110406.

Sohn, K., Berthelot, D., Carlini, N., Zhang, Z., Zhang, H., Raffel, C.A., Cubuk, E.D., Kurakin, A., Li, C.-L., 2020. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. Adv. Neural Inf. Process. Syst. 33, 596–608.

Sun, Y., Chen, C., Wang, Q., Boehm, B., 2017a. Improving missing issue-commit link recovery using positive and unlabeled data. In: Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 147–152.

Sun, Y., Wang, Q., Yang, Y., 2017b. Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance. Inf. Softw. Technol. 84, 33–47.

Tarvainen, A., Valpola, H., 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. Adv. Neural Inf. Process. Syst. 30.

Tian, F., Wang, T., Liang, P., Wang, C., Khan, A.A., Babar, M.A., 2021. The impact of traceability on software maintenance and evolution: A mapping study. J. Softw.: Evol. Process. 33 (10), e2374.

Tian, Y., Zhang, Y., Stol, K.-J., Jiang, L., Liu, H., 2022. What makes a good commit message? In: Proceedings of the 44th International Conference on Software Engineering. ICSE, pp. 2389–2401.

Wong, T.-T., Yeh, P.-Y., 2019. Reliable accuracy estimates from k-fold cross validation. IEEE Trans. Knowl. Data Eng. 32 (8), 1586–1594.

Wu, R., Zhang, H., Kim, S., Cheung, S.-C., 2011. Relink: recovering links between bugs and changes. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. FSE, pp. 15–25.

Xie, Q., Dai, Z., Hovy, E., Luong, T., Le, Q., 2020. Unsupervised data augmentation for consistency training. Adv. Neural Inf. Process. Syst. 33, 6256–6268.

Yang, X., Lo, D., Xia, X., Sun, J., 2017. TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. Inf. Softw. Technol. 87, 206–220.

Yang, X., Song, Z., King, I., Xu, Z., 2022. A survey on deep semi-supervised learning. IEEE Trans. Knowl. Data Eng..

Yu, X., Zhang, H., Li, J., 2021. Text classification method based on semi-supervised transfer learning. In: Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion. QRS-C, IEEE, pp. 388–394.

Zhang, B., Wang, Y., Hou, W., Wu, H., Wang, J., Okumura, M., Shinozaki, T., 2021. Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling. Adv. Neural Inf. Process. Syst. 34, 18408–18419.

Zheng, M., You, S., Huang, L., Wang, F., Qian, C., Xu, C., 2022. Simmatch: Semi-supervised learning with similarity matching. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, pp. 14471–14481.

Zhu, J., Xiao, G., Zheng, Z., Sui, Y., 2022. Enhancing traceability link recovery with unlabeled data. In: Proceedings of the 2022 IEEE 33rd International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 446–457.

Jianfei Zhu is working towards the master' s degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His current research interest is software traceability.

Guanping Xiao is currently an assistant professor at the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He received his Ph.D. degree in software engineering from Beihang University, Beijing, China. His research interests include software reliability, software analysis, SE4AI, and AI4SE.

Zheng Zheng is a professor with Beihang University and the deputy dean with the School of Automation Science and Electrical Engineering. His research work is primarily concerned with software reliability and testing. Recently, He paid more attention on the intelligent software reliability engineering. He has co-authored more than 100 journal and conference publications, including IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Software Engineering, IEEE Transactions on Reliability, IEEE Transactions on Services Computing, JSS and so on. He serves for IEEE PRDC2019, IEEE DASC 2019, IEEE ISSRE 2020, IEEE QRS 2021 as PC Co-Chairs, as well as WoSAR 2019, DeIS 2020 and DeIS 2021 as General Co-Chairs. He is the editor-in-chief of Atlantis Highlights in Engineering (Springer Nature), associate editor of IEEE Transactions on Reliability (2021-), Elsevier KBS (2018-) and Springer IJCIS (2012-) and guest editor of IEEE Transactions on Dependable and Secure Computing (2021). He is IEEE CIS Emerging Technologies TC member.

Yulei Sui is a scientia associate professor at the School of Computer Science and Engineering, the University of New South Wales, Sydney, 2052, Australia. His research interests broadly include program analysis, software engineering, and security. Sui received his Ph.D. degree in software engineering and computer science from the University of New South Wales.