Enhancing Traceability Link Recovery with Unlabeled Data

Jianfei Zhu*, Guanping Xiao*†⋆, Zheng Zheng‡, Yulei Sui§
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
†State Key Laboratory of Novel Software Technology, Nanjing University, China
‡School of Automation Science and Electrical Engineering, Beihang University, China
§School of Computer Science, University of Technology Sydney, Australia
{zjf, gpxiao}@nuaa.edu.cn, zhengz@buaa.edu.cn, yulei.sui@uts.edu.au

Abstract—**Traceability link recovery (TLR) is an important software engineering task for developing trustworthy and reliable software systems. Recently proposed deep learning (DL) models have shown their effectiveness compared to traditional information retrieval-based methods. DL often heavily relies on sufficient labeled data to train the model. However, manually labeling traceability links is time-consuming, labor-intensive, and requires specific knowledge from domain experts. As a result, typically only a small portion of labeled data is accompanied by a large amount of unlabeled data in real-world projects. Our hypothesis is that artifacts are semantically similar if they have the same linked artifact(s).**

This paper presents TraceFUN, a new approach to enhance traceability link recovery with unlabeled data. TraceFUN first measures the similarities between unlabeled and labeled artifacts using two similarity prediction methods (i.e., vector space model and contrastive learning). Then, based on the similarities, newly labeled links are generated between the unlabeled artifacts and the linked objects of the labeled artifacts. Generated links are further used for TLR model training. We have evaluated TraceFUN on three GitHub projects with two state-of-the-art DL models (i.e., Trace BERT and TraceNN). The results show that TraceFUN is effective in terms of a maximum improvement of F1-score up to 21% and 1,088%, respectively for Trace BERT and TraceNN.

Index Terms—**traceability link recovery, unlabeled data, vector space model, contrastive learning**

I. Introduction

Traceability link recovery (TLR) is a software engineering task that recovers links between different types of software artifacts, such as requirements, source code, bug reports, test cases, and user documentation. Traceability plays an important role in software development and maintenance, providing useful support for various software activities, e.g., program comprehension [1], compliance verification [2], change impact analysis [3], and regression analysis of test cases [4].

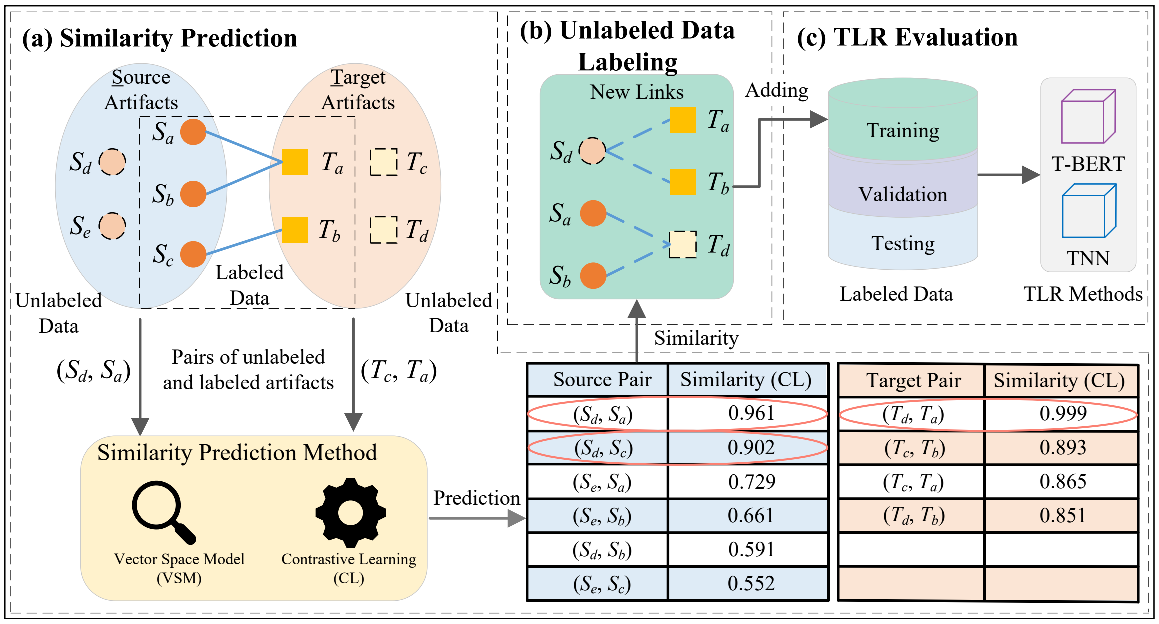
Manually recovering traceability links is time-consuming, labor-intensive, and error-prone. Over the past decades, several TLR approaches have been proposed to ease the burden of developers through automated or semi-automated recovering traceability links [5]–[21]. Recently, deep learning (DL) techniques have been widely adopted in many TLR methods with a substantial performance improvement compared to traditional methods based on information retrieval (IR) [22]–[27].

⋆Corresponding author: Guanping Xiao.

Observations and Insights. DL-based methods often require a large amount of labeled data. However, it is challenging to obtain such data in real-world projects. This is because manual labeling of a large number of ground-truth links from software repositories requires domain knowledge, which is cost-ineffective, especially for large projects. Moreover, even some heuristic rules (e.g., regular expression [28]) are used to retrieve links automatically. Only a small portion of links can be built and many links are still missing [16]. For example, we found that in the Flask dataset collected from GitHub [27], only 752 labeled links related to 1,490 artifacts are established, but there still exist 6,233 unlabeled artifacts.

Typically, such unlabeled artifacts would be excluded from the training dataset for DL-based models. If we can use the abundant source of unlabeled data for model training, it is expected to improve the performance. It is known that multiple source artifacts can link to the same target artifact and vice versa, i.e., many-to-one relationships. Due to the same linked artifact, these source artifacts usually have semantically similar relationships. For example, three duplicate bug reports may relate to one bug-fixing commit. These bug reports described the same failure phenomenon with similar contents. If a newly submitted report also has similar descriptions of the failure, it is highly possible that the report has the same bug-fixing commit.

Our Solution. Inspired by these insights, this paper presents TraceFUN, a traceability links recovery framework enhanced with unlabeled data. Fig. 1 shows the overview of TraceFUN. First, to measure the similarity between unlabeled and labeled artifacts (both can be source artifacts or target artifacts), we introduce two similarity prediction methods, i.e., vector space model (VSM) and contrastive learning (CL), as shown in Fig. 1(a). VSM is a commonly used IR method to predict text similarity [29], while CL is a DL method for classifying similar and dissimilar data samples without labels [30]. By these methods, TraceFUN calculates the similarities between unlabeled and labeled artifacts. Next, new link(s) is (are) generated by connecting the unlabeled artifact to the linked artifact(s) of the labeled one, according to the selected highly similar artifact pairs, as depicted in Fig. 1(b). Last, the newly labeled links are integrated into the training dataset with original labeled data to train DL-based



Source Pair	Similarity (CL)
(S_d, S_a)	0.961
(S_d, S_c)	0.902
(S_e, S_a)	0.729
(S_e, S_b)	0.661
(S_d, S_b)	0.591
(S_e, S_c)	0.552

Target Pair	Similarity (CL)
(T_d, T_a)	0.999
(T_c, T_b)	0.893
(T_c, T_a)	0.865
(T_d, T_b)	0.851

Fig. 1. Overview of TRACEFUN.

TLR methods, as illustrated in Fig. 1(c). We chose two recent state-of-the-art methods, i.e., Trace BERT (T-BERT) [27] and TraceNN (TNN) [26], as the baseline TLR methods. To evaluate TRACEFUN, we compared the performance of these two methods before and after using unlabeled data collected from three open-source GitHub projects, including Flask, Pgcli, and Keras. The impact on TLR performance regarding different similarity prediction methods and sizes of newly labeled data is also investigated.

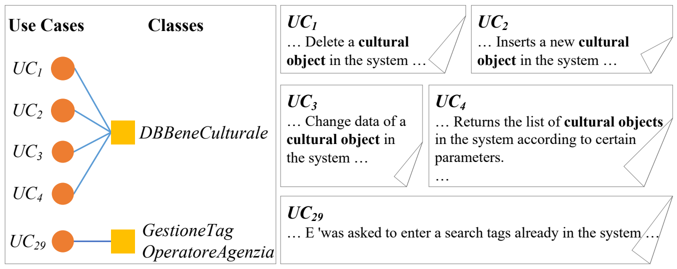
In summary, the paper has the following key contributions:

- To the best of our knowledge, this paper presents the first attempt to use unlabeled data for TLR.
- TRACEFUN, for the first time, introduces VSM and CL methods to measure the similarity between unlabeled and labeled artifacts for generating new training samples.
- We have evaluated TRACEFUN by comparing it with two state-of-the-art methods using 5-fold cross-validation on three GitHub projects. Results show that TRACEFUN boosts T-BERT and TNN with a maximum improvement of F1-score up to 21% and 1,088%, respectively.
- We made the source code of TRACEFUN publicly available at https://github.com/TraceFUN.

II. A Motivating Example

Fig. 2 shows a real-world traceability link example in the *eTOUR* project. Four use cases (i.e., UC_1, UC_2, UC_3, and UC_4) are connected to class *DBBeneCulturale*, which performs the addition, deletion, modification and search operations of the BeanBeneCulturale list. UC_{29} is linked to class *GestioneTagOperatoreAgenzia*, which serves as the common tag management. We can find that all four use cases describe the operations on the same object (i.e., *cultural*), while UC_{29} describes operations related to different objects.

We use VSM to calculate the similarity between two use cases, as shown in TABLE I. The similarities of pairs related



UC_1 … Delete a **cultural object** in the system …

UC_2 … Inserts a new **cultural object** in the system …

UC_3 … Change data of a **cultural object** in the system …

UC_4 … Returns the list of **cultural objects** in the system according to certain parameters. …

UC_{29} … E 'was asked to enter a search tags already in the system …

Fig. 2. Traceability links between use cases and classes in the eTOUR project.

TABLE I
SIMILARITIES BETWEEN USE CASES

Has Common Target Artifact		No Common Target Artifact	
Use Case	**Similarity**	**Use Case**	**Similarity**
(UC_1, UC_3)	0.646	(UC_4, UC_{29})	0.162
(UC_2, UC_3)	0.556	(UC_3, UC_{29})	0.142
(UC_1, UC_2)	0.419	(UC_1, UC_{29})	0.138
(UC_3, UC_4)	0.347	(UC_2, UC_{29})	0.123
(UC_2, UC_4)	0.328		
(UC_1, UC_4)	0.257		

to the four use cases (i.e., UC_1, UC_2, UC_3, and UC_4) are around 0.25 to 0.65, which are significantly higher than those pairs related to UC_{29} (about 0.15). The result is expected, since UC_1, UC_2, UC_3, and UC_4 are connected to the same target class. If one unlabeled use case is similar to the use cases 1 to 4, this use case may have a link to class *DBBeneCulturale*. Therefore, we can leverage such similar relationships to tag the unlabeled data, hence generating more labeled links to training TLR models.

III. Our TraceFUN Approach

There are two major challenges (C) in handling unlabeled data for TLR.

- **C1:** How to measure the similarity between unlabeled and labeled artifacts?

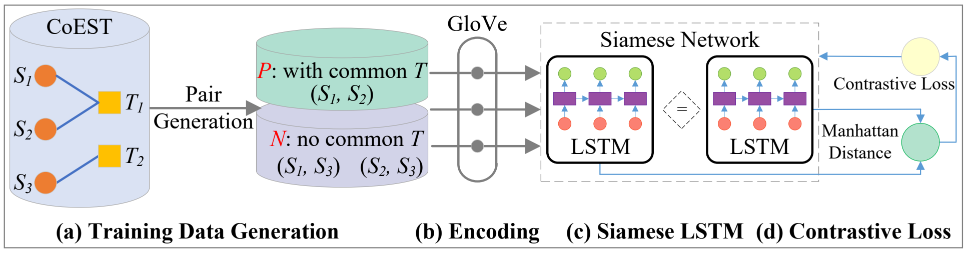


Fig. 3. Detailed structure of contrastive learning in TRACEFUN.

- **C2:** How to label unlabeled data based on the calculated similarities?

To address the two challenges, this section introduces our TRACEFUN, which consists of three parts, i.e., similarity prediction (**C1**), unlabeled data labeling (**C2**), and TLR evaluation, as shown in Fig. 1.

A. Similarity Prediction

TRACEFUN measures the similarity between unlabeled and labeled artifacts from the same type of software artifacts, i.e., source or target artifacts. To obtain all the possible pairs, a Cartesian product calculation for unlabeled and labeled artifacts is performed. The Cartesian product of two sets X and Y, denoted in set theory as $X \times Y$, is the set of all possible ordered pairs:

$$X \times Y = \{(x, y) \mid x \in X \& y \in Y\}, \tag{1}$$

where X is the set of unlabeled artifacts and Y denotes the set of labeled artifacts. For example, after performing Cartesian product calculation on the unlabeled source artifacts $\{S_d, S_e\}$ and the labeled source artifacts $\{S_a, S_b, S_c\}$, the set of all possible pairs $\{(S_d, S_a)$, (S_d, S_b), (S_d, S_c), (S_e, S_a), (S_e, S_b), $(S_e, S_c)\}$ is obtained for further similarity prediction, as depicted in Fig. 1(a). The Cartesian product calculation on the unlabeled and labeled target artifacts is performed in the same way.

Since there are no (similar or dissimilar) labels between unlabeled and labeled artifacts, TRACEFUN introduces two unsupervised similarity prediction methods: vector space model (VSM) and contrastive learning (CL). CL specializes in classifying unlabeled data, which makes similar samples close to each other while dissimilar ones are far apart. For comparison, we use a standard measurement VSM, which is considered the best compared with other traditional similarity prediction methods like LDA and LSI [27]. Note that the similarity prediction methods are not limited to these two models. We can add more methods in TRACEFUN. The details of VSM and CL are described as follows.

1) Vector Space Model: VSM has a document space in which text content is represented as a vector [5]. The document space is represented by an $m \times n$ matrix, where m is the number of terms, and n is the number of documents, aka "term-by-document matrix". An entry $d_{i,j}$ of the matrix represents the weight of the i-th term in the j-th document. In the simplest case, this weight is a boolean value of 1 if the i-th term occurs in the j-th document, or 0 otherwise. A more commonly used measurement is to use the term frequency-inverse document frequency (TF-IDF) method [31]. TF-IDF combines term frequency (TF) and inverse document frequency (IDF). The vector element $d_{i,j}$ is denoted as:

$$d_{i,j} = tf_{i,j} \times idf_i, \tag{2}$$

where $tf_{i,j}$ is the frequency of the i-th term in the j-th document and idf_i is defined as:

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}, \tag{3}$$

where $|D|$ is the total number of documents and $|\{d : t_i \in d\}|$ denotes the number of documents containing term t_i.

We regard all documents as corpus V. A new query Q is represented as a vector in the same way. Then the similarity between query Q and document D_j is as follows:

$$Similarity(D_j, Q) = \frac{\sum_{i=1}^{V} d_{i,j} * q_i}{\sqrt{\sum_{h=1}^{V} (d_{h,j})^2 * \sum_{k=1}^{V} (q_k)^2}}, \tag{4}$$

where $d_{i,j}$ is a one-dimensional vector of the document D_j, and q_i is a one-dimensional vector of the query Q. In TRACEFUN, a query Q represents one unlabeled artifact while D_j denotes the j-th labeled artifact. Both Q and D_j are from the same type of artifacts (i.e., source or target).

2) Contrastive Learning: CL is unsupervised learning to learn a representation function without labels, with the aim to make similar samples closer and dissimilar samples further apart [30].

Fig. 3 shows the process of CL to predict the similarity of the same type of software artifacts. First, due to the absence of labels, some attributes of the data are used to generate pseudo-labels as training data (depicted in Fig. 3(a)). The purpose is to teach the model know which data samples are similar. To obtain such a contrastive representation of artifacts, we use CoEST data [22], [32]–[40], which provides ground-truth traceability links from several projects. Then, the generated pseudo-labeled samples are encoded as vectors through GloVe [41], which are further fed into Siamese long short-term memory (LSTM) neural network [42], as shown in Fig. 3(b) and (c). Last, the contrastive loss function is used to learn a contrastive representation of the data, as shown in Fig. 3(d). Details of each part are described as follows.

Training Data Generation. A key issue in CL is to generate meaningful training data, i.e., a set of paired examples

$\{(x_i, x_i^+)\}_{i=1}^m$ indicates that x_i and x_i^+ are semantically related artifacts. In TRACEFUN, source artifacts with the same linked target artifact are regarded as similar samples (i.e., positive samples). By contrast, dissimilar samples (i.e., negative samples) are generated from source artifacts that are not connected to the same target artifact.

We use a $s \times s$ matrix to record semantically similar relationships between source artifacts, where s is the total number of source artifacts. The element $r_{i,j}$ in the matrix represents the relationship between the i-th source artifact and the j-th source artifact. $r_{i,j} = 1$ represents these two artifacts are semantically similar (i.e., with common target artifacts), while $r_{i,j} = 0$ denotes dissimilar (i.e., no common target artifacts). To generate such a matrix, we traverse the labeled traceability links. The source artifacts linked to the same target artifact are assigned to the same group. The total number of groups is t, which equals the total number of the target artifacts. If there is at least one group has the occurrence of two source artifacts, they are regarded as a positive pair sample, i.e., $r = 1$; if two source artifacts did not occur in the same group, they are regarded as a negative case, i.e., $r = 0$.

Next, we use $itertools.combinations()$, a Python function for Cartesian product [43], to sort the source artifacts in positional order of combinations without duplicate elements, e.g., elements $\{a, b, c, d\}$ generate combinations (a, b), (a, c), (a, d), (b, c), (b, d), and (c, d). Then, the relationship between the two source artifacts is queried from the aforementioned similarity relationship matrix, in order to generate the training data of CL.

Encoding. The text of software artifacts is regarded as an unstructured feature. Each word in the text is assigned a unique index, and the text of the artifact is represented as a sequence of word indices. A word is encoded into a vector representation by GloVe [41] and later forms the word embedding matrix by filling each row of the matrix with the vector in the index order. Next, the word embedding matrix is fed into the Siamese network as weights of the embedding layer, where each word of the sentence finds its vector representation.

Siamese LSTM. Contrastive learning requires a DL model to capture the latent semantic relations of similar and dissimilar data points. In TRACEFUN, we build a Siamese LSTM network to predict similar relationships between software artifacts of the same type. As shown in Fig. 3(c), the network consists of two identical LSTMs.

Assuming that the text of one artifact is transformed into a sequence of word embedding vectors $a = \{w_1, w_2, \ldots, w_{N_T}\}$, where w_i is the embedding representation of the i-th word in this piece of text with a length of N_T. The w_i entering the LSTM at time t is denoted by x_t, as shown in Fig. 4. The output h_t of LSTM at each time step depends on the input x_t at the current time step, the output t_{t-1} at the previous time step (i.e., the short-term memory unit), and the long-term memory C of the network (i.e., the long-term memory unit). LSTM controls how information in a sequence of data enters, stores, and leaves the network using a series of "gates".

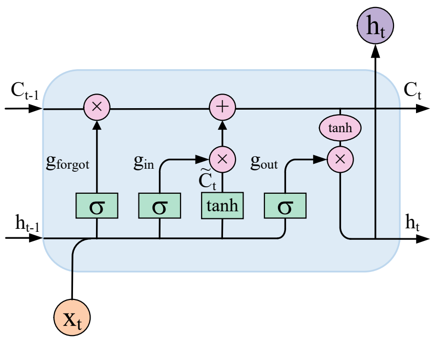


Fig. 4. The structure of LSTM.

First, through the forget gate g_{forgot} decides to forget unnecessary information:

$$g_{forget}(t) = \sigma(W_f * x_t + U_f * h_{t-1}) \tag{5}$$

Then, a new memory update vector $\tilde{C}_t$ is generated according to the short-term memory unit h_{t-1} at the previous time step, and the input x_t at the current time step. LSTM uses the input gate g_{in} to select the information to be remembered and adds it to the last long-term memory unit C_{t-1} filtered by the forget gate, and updates it to a new long-term memory unit C_t:

$$g_{in}(t) = \sigma(W_i * x_t + U_i * h_{t-1}), \tag{6}$$

$$\tilde{C}_t = tanh(W_c * x_t + U_c * h_{t-1}), \tag{7}$$

$$C_t = g_{forget} * C_{t-1} + g_{in} * \tilde{C}_t. \tag{8}$$

Finally, the output gate g_{out} selects the information related to the current task at the current time step to generate the output h_t:

$$g_{out}(t) = \sigma(W_o * x_t + U_o * h_{t-1}), \tag{9}$$

$$h_t = g_{out} * tanh(C_t). \tag{10}$$

In Equations (5)-(10), W and U are weight matrices, σ is the sigmoid function, and $tanh$ is the activation function.

The LSTM outputs the final hidden state to represent the semantic information of the sentence. Therefore, unstructured features are represented as n-dimensional vector h (n is the number of hidden units in the LSTM). In our Siamese LSTM model, we use Manhattan distance to measure the similarity of two vectors, which is defined as:

$$Similarity(h_1, h_2) = exp(- \parallel h_1 - h_2 \parallel_1), \tag{11}$$

where h_1 and h_2 are the vectors of two software artifacts and $exp(.)$ normalizes the distance value between 0 and 1.

Contrastive Loss. Contrastive loss is the training objective, which makes similar samples closer and different samples farther [44]. Contrastive loss is defined as:

$$L = (1-y) * \hat{y}^2 + y * max(0, m - \hat{y})^2, \qquad (12)$$

where m is a hyperparameter that defines the lower bound distance between dissimilar samples. In TRACEFUN, m is set to 1. y and $\hat{y}$ are the true and predicted labels (Manhattan distance) of a pair of two artifacts a_i and a_j, respectively. If two artifacts are not similar (y=0), Equation (12) minimizes their predicted value $\hat{y}$; otherwise it minimizes $max(0, m - \hat{y})^2$, i.e., maximizing their predicted value $\hat{y}$.

B. Unlabeled Data Labeling

After performing similarity prediction on unlabeled and labeled artifacts through each method mentioned above, we have two lists of similarities sorted in descending order, respectively for source and target artifacts. The number (N) of newly generated links from unlabeled data is tuned by the proportion p (e.g., 20% or 50%) of the total number of the original labeled links used for training.

To label the unlabeled artifacts, TRACEFUN traverses pair items in the two similarity lists one by one. For each iteration, SS_i (the similarity of the i-th item in the list of source artifacts) and ST_j (the similarity of the j-th item in the list of target artifacts) are compared to select the higher similar pair, where i and j start from index 0 (i.e., from top to down). For example, if SS_i is greater than ST_j, the i-th source artifact pair (S_p, S_q) is selected for further labeling, where S_p is the unlabeled source artifact with an ID p and S_q is the labeled source artifact with an ID q. New link(s) is (are) generated from S_p to the linked target artifact(s) of S_q. Otherwise, the new link(s) will be generated from unlabeled target artifacts. After generating new labeled links in this iteration, the index i increases by one while the index j remains unchanged for the next comparison. The loop will be terminated when the sum of newly labeled links is larger or equal to N.

Note that since one labeled artifact may have more than one linked object, the cumulative number of generated links in the last iteration may be greater than N. Besides, since two (or more) labeled artifacts may have the same linked artifact. The pairs between one unlabeled artifact and the two labeled artifacts could both have high similarities in the ranked list, thus generating duplicate labeled pairs, which need to be further eliminated.

In this way, TRACEFUN will only select the highly similar source or target artifact pairs to generate links between unlabeled artifacts and the linked object(s) of the labeled ones.

C. TLR Evaluation

As shown in Fig. 1(c), all the newly generated links are added to the training set together with the original labeled links. The validation (if needed) and the testing data samples are only divided from the labeled links. TRACEFUN integrates two recent state-of-the-art DL-based models, i.e., T-BERT [27] and TNN [26]. T-BERT uses the bidirectional language model BERT (Bidirectional Encoder Representations from Transformers) [45], which has richer contextual information. T-BERT includes three implementation stages: pre-training and intermediate training of the T-BERT model using a different source of training data and fine-tuning TLR tasks on real-work projects. TNN uses a tracing network that combines word embedding and recurrent neural network (RNN) to restore the traceability link of software artifacts.

TABLE II
COLLECTED COEST DATASETS FOR CL TRAINING

Project	Source Artifact		Target Artifact		#Links
	Type	#Artifacts	Type	#Artifacts	
Albergate	Requirements	17	Code	55	54
CCHIT	Requirements	116	Code	1,064	587
CM1	High Requirements	22	Low Requirements	53	45
eANCI	Use Cases	140	Code	55	567
EasyClinic	Multi-type	160	Multi-type	160	1,618
EBT	Requirements	41	Code	50	98
eTOUR	Use Cases	58	Code	116	308
GANNT	High Requirements	17	Low Requirements	69	68
HIPAA	Requirements	10	Technical Safeguards	1,891	243
Ice Breaker	Requirements	201	UML Classes	73	457
Infusion Pump	Requirements	126	Components	21	131
iTrust	Requirements	131	Code	367	534
Kiosk	Requirements	178	Processes	178	1,951
SMOS	Use Cases	67	Code	100	1,044
WARC	Requirements	63	Requirements	89	136

TABLE III
COLLECTED DATASETS FOR TRACEFUN EVALUATION

Project	Source Artifact	Target Artifact	#Links
Flask	3,715	4,008	752
Pgcli	1,197	2,189	529
Keras	4,811	5,349	552

TRACEFUN is a flexible framework and capable of using unlabeled data to train any supervised TLR methods (not limited to T-BERT and TNN) that require labeled data.

IV. EXPERIMENT SETUP AND EVALUATION

A. Data Collection and Aggregation

Dataset for CL Training. The training data for CL is collected from 15 datasets provided by CoEST [46], as shown in TABLE II. These datasets include different types of traceability links, e.g., requirements and code, use cases and classes, high-level and low-level requirements. The CoEST data facilitates our exploration of semantically similar relationships between software artifacts.

Dataset for* TRACEFUN *Evaluation. The labeled datasets for evaluating TRACEFUN are collected from [27], including three open-source GitHub projects, i.e., Flask, Pgcli, and Keras, as depicted in TABLE III. Their artifact types are bug reports and bug-fixing commits. Following the evaluation in T-BERT [27], the trace links are from issues to commits, whose granularity is at the file level. Besides, the similarities are calculated between the unlabeled and labeled issues, and also between the unlabeled and labeled commits.

To evaluate TRACEFUN, we used 5-fold stratified cross-validation, i.e., the collected labeled dataset of each project is divided into five-folds, four of which are used for training and the remaining one fold is used for testing. The ratio of the validation set split from the four folds of data samples is 0.2, i.e., 20% of the training set is used for validation.

The number of newly generated links is determined by the proportion p over the number of labeled links in the remaining training set. In our experiments, five proportions, i.e., 5%, 20%, 50%, 80%, and 110% are selected for TRACEFUN's evaluation. TABLE IV shows the number of generated links through two similarity prediction methods, i.e., VSM and CL. Note that since the similarities of source and target pairs are different, the numbers of links generated from source and target artifacts are also different. For example, the ranges of similarities predicted by CL for the top 200 most similar artifacts are [0.68, 1] and [0.99, 1] for the Flask source and target artifacts, respectively.

Data Preprocessing. The collected datasets are cleaned and processed in the following three steps. (1) Word tokenization: the text extracted from artifacts is first divided into a stream of words; (2) stop-word and punctuation removal: stop-words, numbers, and punctuation are further removed; (3) word normalization: word tokens are converted to their lower cases.

B. *Implementation Details*

Settings for VSM. All the source artifacts and target artifacts from the evaluation datasets (TABLE III) are used as the corpus for VSM. The similarity between the two artifacts is calculated by their vectors through Cosine similarity.

Settings for CL. The training parameters of CL are as follows: number of epochs is 100; the batch size is 64; the number of hidden units n in the LSTM network is 50; the ratio of the train over test split is 0.2, and 20% of samples are split from training samples as the validation set.

Settings for TLR Methods. We evaluate two state-of-the-art TLR methods, i.e., T-BERT and TNN, as described in Sec. III-C. The training parameters of T-BERT are as follows: the number of epochs is 400; batch size is 4; learning rate is 0.00004; network architecture is faster SIAMESE. Besides, the training parameters of TNN are as follows: the number of epochs is 1000; batch size is 1; the number of hidden units n is 60; maximum sequence length is 80; learning rate is 0.0001; network model is GRU.

Experiment Environments. The development environment of TRACEFUN is as follows: Python 3.8, TensorFlow 2.6.0, and Keras 2.6.0. All the experiments of TRACEFUN are conducted on the cloud servers equipped with Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz, 62GB memory, and NVIDIA GeForce RTX 2080Ti GPU.

C. TRACEFUN *Evaluation*

Research Questions (RQs). Our evaluation aims to answer the following three RQs:

- **RQ1:** Can TRACEFUN improve TLR performance?
- **RQ2:** What's the impact of different similarity prediction methods used in TRACEFUN on TLR performance?
- **RQ3:** What's the impact of different sizes of newly labeled links generated by TRACEFUN on TLR performance?

Evaluation Metrics. The metrics for evaluating TRACEFUN are as follows:

TABLE IV
NUMBER OF NEWLY LABELED LINKS BY TRACEFUN

Project	Original	p%	VSM		CL	
			Source	Target	Source	Target
Flask	480	5%	6	18	4	20
		20%	6	90	4	92
		50%	6	234	4	236
		80%	18	366	5	379
		110%	43	482	18	510
Pgcli	338	5%	0	16	0	16
		20%	0	67	0	67
		50%	0	169	14	155
		80%	0	270	104	166
		110%	3	368	198	173
Keras	352	5%	6	11	5	12
		20%	48	22	22	48
		50%	137	39	92	84
		80%	235	46	162	119
		110%	333	54	233	154

- **F-scores:** F-scores are the harmonic mean of precision and recall:

$$F_\beta = \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall}, \tag{13}$$

where β is a positive real factor such that recall is considered β times as important as precision. When β=1 (i.e., F1-score), precision and recall are given the same weight. When β=2 (i.e., F2-score), recall is more important than precision. In our experiment, we used F1-score and F2-score for evaluation.

- **Mean Average Precision (MAP):** Average precision (AveP) refers to the average of the maximum precision values at different recall rates. For each source artifact Q, the AveP is calculated according to the position of all n relevant target artifacts in the ranking. MAP is then calculated by averaging the values of AveP. We use MAP@3, i.e., only artifacts ranked ($rank_i$) in the top 3 positions contribute to $AveP$:

$$AveP@3 = \frac{\sum_i^n P}{n}, P = \begin{cases} P@i, & \text{if } rank_i \leqslant 3 \\ 0, & \text{otherwise} \end{cases}, \tag{14}$$

$$MAP@3 = \frac{1}{Q}\sum_{q=1}^{Q} AveP@3. \tag{15}$$

V. RESULTS AND ANALYSIS

This section presents and discusses the evaluation results of TRACEFUN. Tables V, VI, and VII display the performance of T-BERT in terms of F1-score, F2-score, and MAP, respectively, while those results for TNN are given in Tables VIII, IX, and X, respectively.

A. *RQ1: Can* TRACEFUN *improve TLR performance?*

RQ1 aims to investigate whether our TRACEFUN can improve the performance of TLR. We averaged the results obtained from 5-fold stratified cross-validation and calculated the improvement, to compare the performance of T-BERT and

TABLE V
F1-SCORE OF T-BERT TRAINED WITH NEWLY LABELED DATA GENERATED BY TRACEFUN (VSM AND CL), AND RANDOM SELECTION

Project	Fold	Original	VSM (5%)	CL (5%)	VSM (20%)	CL (20%)	VSM (50%)	CL (50%)	VSM (80%)	CL (80%)	VSM (110%)	CL (110%)	Random (50%)
Flask	1	0.632	0.648	0.654	0.672	0.684	0.713	0.706	0.789	**0.813**	0.783	0.783	0.582
	2	0.658	0.699	0.678	0.675	0.687	0.727	0.727	0.762	0.753	**0.772**	0.771	0.589
	3	0.617	0.613	0.598	0.669	0.642	0.701	0.700	0.699	0.701	**0.767**	0.728	0.506
	4	0.679	0.713	0.667	0.698	0.679	0.725	0.721	**0.789**	0.752	0.788	0.751	0.629
	5	0.636	0.660	0.672	0.694	0.664	0.725	0.713	0.756	0.768	0.779	**0.783**	0.572
	Avg. (Impro.)	0.644	0.667 (4%)	0.654 (2%)	0.682 (6%)	0.671 (4%)	0.718 (12%)	0.713 (11%)	0.759 (18%)	0.757 (18%)	**0.778 (21%)**	0.763 (19%)	0.576 (-11%)
Pgcli	1	0.735	0.715	0.720	0.758	0.790	0.760	0.767	**0.810**	0.751	0.806	0.720	0.620
	2	0.733	0.725	0.714	0.755	0.753	0.747	0.756	0.805	0.687	**0.829**	0.677	0.629
	3	0.757	0.730	0.794	0.794	0.769	0.761	0.792	**0.819**	0.736	0.778	0.742	0.651
	4	0.693	0.735	0.681	0.726	0.697	0.751	0.728	0.798	0.685	**0.821**	0.663	0.640
	5	0.767	0.780	0.759	0.802	0.777	0.808	0.800	0.816	0.709	**0.851**	0.684	0.648
	Avg. (Impro.)	0.737	0.737 (0%)	0.734 (0%)	0.767 (4%)	0.757 (3%)	0.765 (4%)	0.769 (4%)	0.810 (10%)	0.714 (-3%)	**0.817 (11%)**	0.697 (-5%)	0.638 (-13%)
Keras	1	0.938	0.916	0.914	**0.940**	0.927	0.877	0.914	0.871	0.914	0.853	0.883	0.892
	2	**0.950**	0.941	0.945	0.900	0.932	0.874	0.922	0.883	0.877	0.878	0.858	0.890
	3	0.953	0.940	**0.959**	0.943	0.943	0.938	0.935	0.916	0.911	0.841	0.882	0.906
	4	0.926	0.932	0.927	**0.943**	0.925	0.941	0.897	0.887	0.912	0.859	0.912	0.827
	5	0.960	**0.964**	0.954	0.955	0.963	0.919	0.933	0.920	0.935	0.900	0.910	0.895
	Avg. (Impro.)	**0.945**	0.939 (-1%)	0.940 (-1%)	0.936 (-1%)	0.938 (-1%)	0.910 (-4%)	0.920 (-3%)	0.895 (-5%)	0.910 (-4%)	0.866 (-8%)	0.889 (-6%)	0.882 (-7%)

TABLE VI
F2-SCORE OF T-BERT TRAINED WITH NEWLY LABELED DATA GENERATED BY TRACEFUN (VSM AND CL), AND RANDOM SELECTION

Project	Fold	Original	VSM (5%)	CL (5%)	VSM (20%)	CL (20%)	VSM (50%)	CL (50%)	VSM (80%)	CL (80%)	VSM (110%)	CL (110%)	Random (50%)
Flask	1	0.650	0.612	0.638	0.639	0.659	0.668	0.690	0.760	**0.787**	0.781	0.774	0.562
	2	0.683	0.675	0.680	0.702	0.700	0.744	0.746	0.776	0.768	0.779	**0.788**	0.615
	3	0.604	0.603	0.580	0.638	0.623	0.684	0.675	0.707	0.711	**0.795**	0.740	0.488
	4	0.665	0.696	0.669	0.709	0.698	0.719	0.721	**0.805**	0.751	0.797	0.771	0.628
	5	0.663	0.683	0.694	0.705	0.681	0.724	0.722	**0.797**	0.791	0.791	0.777	0.557
	Avg. (Impro.)	0.653	0.654 (0%)	0.652 (0%)	0.679 (4%)	0.672 (3%)	0.708 (8%)	0.711 (9%)	0.769 (18%)	0.762 (17%)	**0.789 (21%)**	0.770 (18%)	0.570 (-13%)
Pgcli	1	0.747	0.749	0.750	0.769	0.790	0.782	0.777	**0.826**	0.730	0.822	0.695	0.627
	2	0.757	0.755	0.742	0.756	0.743	0.782	0.783	0.839	0.680	**0.850**	0.677	0.641
	3	0.808	0.780	0.821	0.814	0.810	0.806	0.807	**0.838**	0.721	0.810	0.744	0.672
	4	0.701	0.724	0.711	0.748	0.719	0.758	0.749	0.788	0.712	**0.842**	0.661	0.620
	5	0.777	0.812	0.784	0.810	0.802	0.809	0.807	0.822	0.729	**0.856**	0.702	0.668
	Avg. (Impro.)	0.758	0.764 (1%)	0.762 (0%)	0.779 (3%)	0.773 (2%)	0.787 (4%)	0.785 (4%)	0.823 (9%)	0.714 (-6%)	**0.836 (10%)**	0.696 (-8%)	0.646 (-15%)
Keras	1	**0.943**	0.927	0.938	0.936	0.941	0.906	0.924	0.895	0.884	0.886	0.868	0.897
	2	0.944	**0.946**	0.942	0.892	0.924	0.874	0.930	0.886	0.892	0.889	0.893	0.880
	3	0.937	0.932	**0.951**	0.927	0.938	0.915	0.925	0.919	0.897	0.855	0.870	0.887
	4	0.948	0.950	0.954	**0.961**	0.946	0.939	0.918	0.897	0.905	0.871	0.903	0.833
	5	0.967	**0.973**	0.959	0.957	0.966	0.919	0.950	0.940	0.939	0.916	0.927	0.893
	Avg. (Impro.)	0.948	0.946 (0%)	**0.949 (0%)**	0.935 (-1%)	0.943 (-1%)	0.911 (-4%)	0.929 (-2%)	0.907 (-4%)	0.903 (-5%)	0.883 (-7%)	0.892 (-6%)	0.878 (-7%)

TNN before and after adding newly labeled data via TRACEFUN (i.e., VSM and CL). In addition, the performance trained by randomly generating labeled links is also investigated. Note, for the random selection, we combine the pair lists of source and target artifacts together, and then randomly shuffle one pair from them without considering the similarities. New links are generated in the same way as described in Sec. III-B. This process stops until the number of newly labeled links is larger or equal to a given size (i.e., 50% of the original labeled data used for training).

It can be seen from Table V to Table X that, for the Flask and Pgcli datasets, the scores of all the three evaluation metrics (i.e., F1-score, F2-score, and MAP) of T-BERT and TNN significantly improve after adding newly labeled links by VSM and CL (i.e., TRACEFUN). For T-BERT, the maximum improvements of F1-score, F2-score, and MAP of the Flask dataset, are 21%, 21%, and 19%, respectively, while those results of the Pgcli dataset are 11%, 10%, and 11%, respectively. Similarly, the performance of TNN on these two datasets also has a dramatic improvement after adding newly labeled links generated by TRACEFUN. For example, the maximum improvements in terms of F1-score, F2-score, and MAP are 1,088%, 555%, 1,091%, and 719%, 322%, 733%, respectively for the Flask and Pgcli datasets.

In addition, for the Keras dataset, the performance improvements of T-BERT and TNN after adding new links by TRACEFUN are different. For T-BERT, the evaluation scores are slightly degraded compared to the results obtained from the original training set. For example, the F1-score of Keras using the original labeled dataset is 0.945, while the best result by TRACEFUN is 0.940. However, for TNN, the performance is increased with the best improvements of 109%, 41%, and 146%, respectively for F1-score, F2-score, and MAP, although such improvements are lower than those of the Flask and Pgcli datasets, as shown in Tables VIII, IX, and X. Note, similar to the results reported in [27], the performance of TNN using original labeled data is extremely low, e.g., F1-scores of TNN on the Flask, Pgcli, and Keras datasets are 0.033, 0.036, and 0.032, respectively. This is because TNN requires a large amount of labeled data to achieve good performance [27]. The original labeled data from the three datasets is quite small, thus causing poor performance.

Furthermore, for random selection, except for TNN on the Pgcli datasets regarding F1-score and MAP, the scores of all

TABLE VII

MAP of T-BERT Trained with Newly Labeled Data Generated by TraceFUN (VSM and CL), and Random Selection

Project	Fold	Original	VSM (5%)	CL (5%)	VSM (20%)	CL (20%)	VSM (50%)	CL (50%)	VSM (80%)	CL (80%)	VSM (110%)	CL (110%)	Random (50%)
Flask	1	0.713	0.731	0.720	0.732	0.772	0.798	0.779	0.836	0.841	**0.863**	0.860	0.630
	2	0.733	0.756	0.743	0.768	0.786	0.800	0.832	0.845	0.849	0.848	**0.859**	0.657
	3	0.688	0.684	0.672	0.737	0.693	0.758	0.773	0.787	0.812	**0.861**	0.849	0.558
	4	0.741	0.769	0.768	0.784	0.797	0.800	0.820	0.862	0.852	**0.880**	0.854	0.687
	5	0.756	0.760	0.790	0.790	0.767	0.818	0.814	0.846	0.869	**0.883**	0.863	0.639
	Avg. (Impro.)	0.726	0.740 (2%)	0.739 (2%)	0.762 (5%)	0.763 (5%)	0.795 (9%)	0.804 (11%)	0.835 (15%)	0.845 (16%)	**0.867 (19%)**	0.857 (18%)	0.634 (-13%)
Pgcli	1	0.803	0.819	0.797	0.833	0.836	0.847	0.825	0.869	0.797	**0.890**	0.770	0.690
	2	0.803	0.846	0.813	0.810	0.846	0.863	0.860	0.895	0.770	**0.918**	0.775	0.695
	3	0.849	0.847	0.862	0.863	0.869	0.865	0.866	**0.898**	0.792	0.869	0.846	0.734
	4	0.737	0.767	0.789	0.775	0.802	0.830	0.803	0.851	0.781	**0.885**	0.745	0.656
	5	0.865	0.876	0.873	0.894	0.894	0.884	0.887	0.905	0.873	**0.938**	0.821	0.748
	Avg. (Impro.)	0.811	0.831 (2%)	0.827 (2%)	0.835 (3%)	0.849 (5%)	0.858 (6%)	0.848 (5%)	0.884 (9%)	0.803 (-1%)	**0.900 (11%)**	0.791 (-2%)	0.705 (-13%)
Keras	1	0.973	0.962	**0.977**	0.965	0.949	0.973	0.935	0.955	0.946	0.940	0.902	0.940
	2	0.967	0.964	**0.973**	0.935	**0.973**	0.958	0.968	0.946	0.946	0.934	0.967	0.917
	3	0.948	**0.973**	0.961	0.955	**0.973**	0.945	0.948	0.959	0.933	0.930	0.895	0.950
	4	**0.986**	0.973	0.964	0.977	0.959	0.973	0.950	0.959	0.955	0.939	0.944	0.905
	5	0.977	0.977	0.977	0.977	0.982	0.962	**0.986**	0.982	0.971	0.971	0.973	0.964
	Avg. (Impro.)	**0.970**	**0.970 (0%)**	**0.970 (0%)**	0.962 (-1%)	0.967 (0%)	0.926 (-1%)	0.957 (-1%)	0.960 (-1%)	0.950 (-3%)	0.943 (-3%)	0.936 (-3%)	0.935 (-4%)

TABLE VIII

F1-score of TNN Trained with Newly Labeled Data Generated by TraceFUN (VSM and CL), and Random Selection

Project	Fold	Original	VSM (5%)	CL (5%)	VSM (20%)	CL (20%)	VSM (50%)	CL (50%)	VSM (80%)	CL (80%)	VSM (110%)	CL (110%)	Random (50%)
Flask	1	0.031	0.021	0.037	0.060	0.061	0.130	0.157	0.232	0.310	0.287	**0.398**	0.027
	2	0.025	0.051	0.025	0.071	0.074	0.182	0.181	0.211	0.347	0.331	**0.384**	0.020
	3	0.043	0.025	0.023	0.048	0.046	0.115	0.102	0.153	0.292	0.244	**0.346**	0.015
	4	0.028	0.026	0.037	0.090	0.088	0.187	0.189	0.253	0.355	0.357	**0.423**	0.026
	5	0.036	0.032	0.054	0.046	0.052	0.130	0.190	0.202	0.349	0.313	**0.410**	0.020
	Avg. (Impro.)	0.033	0.031 (-6%)	0.035 (7%)	0.063 (91%)	0.064 (95%)	0.149 (351%)	0.164 (396%)	0.210 (537%)	0.331 (902%)	0.306 (828%)	**0.392 (1,088%)**	0.022 (-35%)
Pgcli	1	0.036	0.042	0.038	0.087	0.081	0.142	0.231	0.255	0.260	**0.380**	0.252	0.031
	2	0.035	0.030	0.033	0.058	0.048	0.168	0.154	0.245	0.125	**0.288**	0.178	0.038
	3	0.039	0.047	0.049	0.100	0.083	0.164	0.169	0.240	0.176	**0.292**	0.215	0.050
	4	0.031	0.032	0.033	0.053	0.055	0.091	0.154	0.184	0.165	**0.238**	0.146	0.035
	5	0.040	0.083	0.068	0.110	0.105	0.174	0.198	0.241	0.186	**0.277**	0.222	0.040
	Avg. (Impro.)	0.036	0.047 (30%)	0.044 (23%)	0.082 (127%)	0.074 (107%)	0.148 (311%)	0.181 (403%)	0.233 (547%)	0.182 (407%)	**0.295 (719%)**	0.203 (463%)	0.039 (8%)
Keras	1	0.029	0.033	0.041	0.064	0.040	0.088	0.043	0.090	0.041	**0.091**	0.039	0.029
	2	0.041	0.026	**0.048**	0.030	0.029	0.037	0.039	0.031	0.022	0.041	0.036	0.033
	3	0.027	0.078	0.026	0.085	0.031	0.108	0.037	**0.109**	0.030	0.104	0.045	0.034
	4	0.029	0.035	0.043	0.030	**0.062**	0.045	0.059	0.052	0.052	0.053	0.058	0.023
	5	0.033	0.031	0.031	0.051	0.028	**0.052**	0.034	0.025	0.021	0.046	0.027	0.021
	Avg. (Impro.)	0.032	0.041 (27%)	0.038 (18%)	0.052 (63%)	0.038 (19%)	0.066 (106%)	0.042 (33%)	0.061 (92%)	0.033 (4%)	**0.067 (109%)**	0.041 (28%)	0.028 (-13%)

evaluation metrics of both T-BERT and TNN are worse than those results trained by the original labeled data. For example, the F1-score, F2-score, and MAP of T-BERT on the Flask dataset dropped by 11%, 13%, and 13%, respectively. The result is expected because random selection has a better chance of introducing mislabeled links, which could negatively impact the model training.

Answer to RQ1: TraceFUN can significantly improve TLR performance. TraceFUN is able to capture the semantically similar relationships between unlabeled and labeled artifacts, thereby generating effective newly labeled links for TLR model training.

B. RQ2: What's the impact of different similarity prediction methods used in TraceFUN *on TLR performance?*

In this RQ, we evaluate the impact of VSM and CL used in TraceFUN on TLR performance. For T-BERT, as illustrated in Tables V, VI, and VII, VSM is better at measuring the similarities between unlabeled and labeled artifacts than CL. For example, the best improvement of the F1-score on the Flask dataset by VSM is 21%, while that by CL is 19%. On the Pgcli dataset, the discrepancy in the performance improvement between VSM and CL is significantly large, e.g., the best improvements in terms of F1-score are 11% and 4%, respectively for VSM and CL.

However, for TNN shown in Tables VIII, IX, and X, using CL is better than using VSM on the Flask dataset. The best performance improvement of the F1-score by CL is 1,088%, while VSM only achieves an improvement of 828%. Such a situation is changed on the Pgcli dataset. VSM obtains the best improvement of 719% of the F1-score, while CL only improves the metric by 463% in the best case.

Answer to RQ2: The performance improvements by VSM and CL used in TraceFUN are different regarding different TLR methods and datasets. It is necessary to select a suitable similarity prediction method according to specific TLR methods and datasets to improve the performance better. For the TLR task on issues and commits, users are suggested to use VSM and CL.

C. RQ3: What's the impact of different sizes of newly labeled links generated by TraceFUN *on TLR performance?*

In this experiment, we generate different sizes of newly labeled links, i.e., 5%, 20%, 50%, 80%, and 110% of the original labeled links used for training, to evaluate their impact on TLR performance.

TABLE IX

F2-SCORE OF TNN TRAINED WITH NEWLY LABELED DATA GENERATED BY TRACEFUN (VSM AND CL), AND RANDOM SELECTION

Project	Fold	Original	VSM (5%)	CL (5%)	VSM (20%)	CL (20%)	VSM (50%)	CL (50%)	VSM (80%)	CL (80%)	VSM (110%)	CL (110%)	Random (50%)
Flask	1	0.050	0.040	0.059	0.071	0.066	0.125	0.162	0.198	0.250	0.234	**0.334**	0.038
	2	0.048	0.048	0.049	0.072	0.077	0.152	0.175	0.197	0.282	0.291	**0.297**	0.040
	3	0.046	0.038	0.042	0.057	0.063	0.120	0.108	0.156	0.216	0.220	**0.252**	0.036
	4	0.043	0.043	0.045	0.069	0.072	0.161	0.161	0.217	0.298	0.313	**0.353**	0.048
	5	0.051	0.051	0.054	0.075	0.068	0.125	0.166	0.206	0.279	0.278	**0.335**	0.043
	Avg. (Impro.)	0.048	0.044 (-8%)	0.050 (4%)	0.069 (43%)	0.069 (44%)	0.137 (185%)	0.154 (222%)	0.195 (306%)	0.265 (452%)	0.267 (457%)	**0.314 (555%)**	0.041 (-15%)
Pgcli	1	0.059	0.071	0.066	0.110	0.100	0.140	0.195	0.220	0.215	**0.295**	0.204	0.055
	2	0.067	0.055	0.065	0.059	0.066	0.138	0.126	0.194	0.131	**0.255**	0.145	0.053
	3	0.050	0.067	0.062	0.077	0.097	0.149	0.152	0.198	0.193	**0.232**	0.169	0.063
	4	0.054	0.060	0.053	0.070	0.073	0.119	0.123	0.186	0.142	**0.231**	0.129	0.052
	5	0.064	0.064	0.079	0.107	0.099	0.166	0.178	0.192	0.176	**0.232**	0.171	0.056
	Avg. (Impro.)	0.059	0.063 (7%)	0.065 (10%)	0.085 (43%)	0.087 (47%)	0.142 (141%)	0.155 (162%)	0.198 (236%)	0.171 (191%)	**0.249 (322%)**	0.164 (177%)	0.056 (-5%)
Keras	1	0.047	0.062	0.052	0.061	0.074	0.072	0.068	0.077	0.058	**0.087**	0.066	0.054
	2	0.065	0.054	**0.071**	0.059	0.053	0.054	0.047	0.066	0.049	0.068	0.054	0.061
	3	0.054	0.073	0.055	0.081	0.051	**0.104**	0.053	0.078	0.054	0.080	0.050	0.055
	4	0.050	0.050	0.061	0.053	**0.091**	0.074	0.050	0.061	0.051	0.073	0.052	0.046
	5	0.048	0.056	0.047	0.049	0.050	0.054	0.046	0.047	0.044	**0.065**	0.044	0.049
	Avg. (Impro.)	0.053	0.059 (11%)	0.057 (8%)	0.061 (14%)	0.064 (20%)	0.072 (35%)	0.053 (0%)	0.066 (24%)	0.051 (-3%)	**0.075 (41%)**	0.053 (0%)	0.053 (0%)

TABLE X

MAP OF TNN TRAINED WITH NEWLY LABELED DATA GENERATED BY TRACEFUN (VSM AND CL), AND RANDOM SELECTION

Project	Fold	Original	VSM (5%)	CL (5%)	VSM (20%)	CL (20%)	VSM (50%)	CL (50%)	VSM (80%)	CL (80%)	VSM (110%)	CL (110%)	Random (50%)
Flask	1	0.031	0.020	0.032	0.063	0.049	0.118	0.155	0.198	0.257	0.243	**0.326**	0.025
	2	0.010	0.034	0.017	0.052	0.085	0.147	0.163	0.178	0.290	0.286	**0.321**	0.009
	3	0.040	0.022	0.026	0.048	0.053	0.114	0.099	0.149	0.238	0.220	**0.270**	0.013
	4	0.024	0.013	0.039	0.069	0.072	0.160	0.167	0.204	0.302	0.318	**0.354**	0.026
	5	0.030	0.029	0.058	0.047	0.053	0.131	0.171	0.216	0.301	0.281	**0.337**	0.018
	Avg. (Impro.)	0.027	0.024 (-13%)	0.034 (27%)	0.056 (107%)	0.062 (131%)	0.134 (396%)	0.151 (459%)	0.189 (600%)	0.278 (928%)	0.270 (899%)	**0.322 (1,091%)**	0.018 (-33%)
Pgcli	1	0.038	0.042	0.031	0.082	0.099	0.137	0.222	0.225	0.201	**0.310**	0.208	0.041
	2	0.041	0.014	0.027	0.042	0.044	0.135	0.121	0.186	0.123	**0.236**	0.149	0.050
	3	0.022	0.052	0.038	0.075	0.085	0.140	0.149	0.215	0.181	**0.250**	0.167	0.028
	4	0.017	0.014	0.041	0.047	0.058	0.090	0.112	0.168	0.134	**0.206**	0.132	0.039
	5	0.032	0.062	0.070	0.097	0.097	0.156	0.195	0.210	0.170	**0.248**	0.183	0.032
	Avg. (Impro.)	0.030	0.037 (23%)	0.041 (38%)	0.069 (129%)	0.077 (155%)	0.132 (339%)	0.160 (433%)	0.201 (569%)	0.162 (439%)	**0.250 (733%)**	0.168 (459%)	0.038 (27%)
Keras	1	0.027	0.014	0.026	0.038	0.038	0.063	0.050	0.050	0.035	**0.071**	0.039	0.027
	2	0.021	0.017	0.038	0.035	0.029	0.033	0.038	0.030	0.018	0.044	**0.045**	0.026
	3	0.023	0.067	0.026	0.062	0.036	0.085	0.032	**0.086**	0.023	0.076	0.036	0.021
	4	0.021	0.039	0.039	0.033	**0.080**	0.041	0.036	0.058	0.056	0.052	0.048	0.015
	5	0.026	0.017	0.024	0.036	0.023	**0.062**	0.044	0.023	0.003	0.052	0.027	0.009
	Avg. (Impro.)	0.024	0.031 (28%)	0.031 (28%)	0.041 (70%)	0.041 (72%)	0.057 (137%)	0.040 (67%)	0.049 (106%)	0.027 (13%)	**0.059 (146%)**	0.039 (63%)	0.020 (-18%)

For T-BERT, as depicted in Tables V, VI, and VII, regarding the Flask dataset, T-BERT's performance goes up along with the increasing proportion up to 110% of newly labeled links by VSM and CL, i.e., the more links used the better the performance. For the Pgcli dataset, VSM presents the same trend, i.e., the performance can be improved by adding more newly labeled links. However, the performance achieved by CL first increases and then decreases with the increasing size of new links. The best proportion for CL on the Pgcli dataset is 50%. For TNN, as shown in Tables VIII, IX, and X, the performances on the Flask and Pgcli datasets all increase with the increasing number of newly labeled links.

In addition, the performance is worse in some proportions after adding newly labeled links by TRACEFUN than the original one. For example, T-BERT's performance on the Pgcli dataset obtained by CL drops at 80% and 110%. For the Keras dataset, regardless of the similarity prediction method used, the performance decreases with the increasing proportion of newly labeled links. The best performance achieved by VSM and CL is 5%, where the performance is slightly lower than the original one. Such performance degradation that appeared on the Keras dataset may be caused by the following reason. The original labeled data is sufficient to train a good T-BERT model, i.e., F1-score, F2-score, and MAP have already achieved up to 0.945, 0.948, and 0.970, respectively. There is little room for improvement.

Answer to RQ3: Generally, TLR performance can be improved by adding more labeled links via TRACEFUN. However, for different TLR methods and datasets, the size of newly labeled links greatly impacts the performance. Therefore, it is necessary to fine-tune the size of new links labeled by TRACEFUN to obtain a better result according to specific TLR methods and datasets.

D. *Limitations of* TRACEFUN

TRACEFUN has two limitations. First, the optimal number of newly added links cannot be determined at one time. For example, the performance achieved by CL first increases and then decreases with the increasing size of new links for the Pgcli dataset. Tuning the proportion of newly added links is necessary.

Second, TRACEFUN may cause performance degradation. For some projects with relatively well-labeled data, TRACEFUN may have performance degradation, as the noise data would be introduced along with the newly labeled links from unlabeled data, e.g., the Keras dataset.

VI. Threats to Validity

Threats to Internal Validity. The used similarity prediction method in TraceFUN is the main threat to internal validity. To reduce this threat, we introduced two different methods, i.e., VSM and CL, to measure the similarity between unlabeled and labeled artifacts. For training the CL model, we only used natural language text extracted from source artifacts as features embedded by Word2Vec-like techniques (e.g., GloVe [41]) to capture the semantically similar relationship. The similarity prediction can be improved by using code embedding techniques [47] (e.g., code2seq [48], code2vec [49], and Flow2Vec [50]) as features for artifacts that majorly contain source code. As a result, TraceFUN's performance can be further improved because more similar artifact pairs can be captured to generate new links to training TLR models.

Threats to External Validity. The generalization of TraceFUN is the main threat to external validity. First, the integrated TLR methods in TraceFUN and the evaluated datasets of traceability links are threats to external validity. To reduce this threat, we select two state-of-the-art DL-based TLR methods, i.e., T-BERT [27] and TNN [26], in which T-BERT is the latest method and shows excellent performance on small datasets compared to TNN. Besides, to evaluate T-BERT and TNN, we use the same datasets, i.e., Flask, Pgcli, and Keras, provided by [27] for evaluation. TraceFUN is applicable for any supervised TLR methods that need labeled data. In the future, we will integrate more TLR methods and use more types of traceability link datasets to evaluate TraceFUN.

Threats to Construct Validity. The choice of evaluation metrics for predictive performance can pose a threat to construct validity. To reduce this threat, we use F-scores and MAP, which are the same evaluation metrics used in T-BERT [27].

VII. Related Work

Deep Learning for TLR. Over the past decades, researchers have proposed various TLR methods [5]–[21]. Recently, deep learning techniques have been applied to TLR given their good performance [22]–[27]. Mills *et al.* [22] propose the TRAIL method, which uses the traceability links to train machine learning models to verify whether the links between new software artifacts are correct. Ruan *et al.* [24] propose DeepLink for recovering links between issues and commits using word embedding and RNN. The semantic relationships between issues and commits are learned. Guo *et al.* [26] propose TraceNN (TNN) that uses word embedding to solve the problem of vocabulary mismatch. The semantic relationship between words is represented by the linear relationship between word embedding vectors. Vectors of artifacts are then fed into an RNN network to predict traceability links between software artifacts. Lin *et al.* introduce Trace BERT (T-BERT) [27], a TLR framework combined with the BERT model. T-BERT effectively transfers knowledge learned from code search problems into TLR using pre-trained language models and transfer learning. The problem of insufficient pre-training data is alleviated, and the accuracy of restored links is improved. To the best of our knowledge, our work first investigates the method for leveraging readily available unlabeled data via generating newly labeled links to further enhance TLR model training.

Contrastive Learning. Contrastive learning (CL) is an approach to formulate the task of finding similar and dissimilar data samples. It can classify data into similar and different without labels. CL has been successfully applied in computer vision [51]–[53]. Chen *et al.* [30] propose the SimCLR method to learn representations by maximizing the consistency between different augmented views of the same data example through a contrastive loss in the latent space, outperforming previous self-supervised and semi-supervised learning methods by a large margin. Recently, CL also has some applications in software engineering. Bui *et al.* [54] propose the Corder method, which identifies similar and different code fragments through CL, and uses a large amount of unlabeled source code data to train a neural network to identify semantically equivalent code fragments. Corder significantly outperforms models without CL in the code retrieval task. Cheng *et al.* [55] propose ContraFlow, a selective yet precise contrastive value-flow embedding approach to statically detect software vulnerabilities. Our paper introduces CL to the field of TLR for the first time. We used CL to capture the semantically similar relationship between unlabeled and labeled artifacts, aiming to supplement the limited training data with abundant unlabeled data by generating newly labeled links.

VIII. Conclusion

This paper presents TraceFUN, a novel TLR framework enhanced with unlabeled data. In TraceFUN, the semantically similar relationships between unlabeled and labeled artifacts are first predicted by VSM and CL. Then, highly similar artifact pairs are selected to generate links between the unlabeled artifacts and the linked artifacts of the labeled ones. These newly labeled links are later added to the training set with the original labeled data for TLR model training. We have comparatively evaluated TraceFUN with two state-of-the-art TLR models (i.e., T-BERT and TNN) on three GitHub projects, including Flask, Pgcli, and Keras. We have also investigated the impact of different similarity prediction methods used in TraceFUN and sizes of newly generated labeled links on TLR performance. Our results show that TraceFUN effectively improves TLR performance by leveraging unlabeled data. Our source code and data are available at https://github.com/TraceFUN.

Acknowledgment

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, Open Research Fund of State Key Laboratory of Novel Software Technology (Nanjing University) under Grant KFKT2020B20, and Australian Research Council under Grant DP210101348.

REFERENCES

[1] M. Grechanik, K. S. McKinley, and D. E. Perry, "Recovering and using use-case-diagram-to-source-code traceability links," in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*, 2007, pp. 95–104.

[2] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 58–93, 2001.

[3] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software*, vol. 80, no. 6, pp. 918–934, 2007.

[4] L. Naslavsky and D. J. Richardson, "Using traceability to support model-based regression testing," in *Proceedings of the 22th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2007, pp. 567–570.

[5] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[6] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering (ICSE)*. IEEE, 2003, pp. 125–135.

[7] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, pp. 13–es, 2007.

[8] X. Chen, "Extraction and visualization of traceability relationships between documents and source code," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2010, pp. 505–510.

[9] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2011, pp. 223–232.

[10] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 47–57.

[11] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 725–741, 2012.

[12] N. Ali, F. Jaafar, and A. E. Hassan, "Leveraging historical co-change information for requirements traceability," in *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 361–370.

[13] J. Guo, N. Monaikul, C. Plepel, and J. Cleland-Huang, "Towards an intelligent domain-specific traceability solution," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 2014, pp. 755–766.

[14] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "When and how using structural information to improve ir-based traceability recovery," in *Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 2013, pp. 199–208.

[15] A. Panichella, A. De Lucia, and A. Zaidman, "Adaptive user feedback for ir-based traceability recovery," in *Proceedings of the 8th IEEE/ACM International Symposium on Software and Systems Traceability (SST)*. IEEE, 2015, pp. 15–21.

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

[17] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "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 (ESEC/FSE)*, 2011, pp. 15–25.

[18] A. Sureka, S. Lal, and L. Agarwal, "Applying fellegi-sunter (fs) model for traceability link recovery between bug databases and version archives," in *Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2011, pp. 146–153.

[19] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 1–11.

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

[21] Y. Sun, Q. Wang, and Y. Yang, "Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance," *Information and Software Technology*, vol. 84, pp. 33–47, 2017.

[22] C. Mills, J. Escobar-Avila, and S. Haiduc, "Automatic traceability maintenance via machine learning classification," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 369–380.

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

[24] H. Ruan, B. Chen, X. Peng, and W. Zhao, "Deeplink: Recovering issue-commit links based on deep learning," *Journal of Systems and Software*, vol. 158, p. 110406, 2019.

[25] A. C. Marcén, R. Lapeña, O. Pastor, and C. Cetina, "Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case," *Journal of Systems and Software*, vol. 163, p. 110519, 2020.

[26] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *Proceedings of the 39th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 3–14.

[27] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," in *Proceedings of the 43th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 324–335.

[28] A. Bachmann and A. Bernstein, "Software process data quality and characteristics: a historical view on open and closed source projects," in *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, 2009, pp. 119–128.

[29] D. Harman, "Ranking algorithms," in *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Inc., 1992, pp. 363–392.

[30] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*. PMLR, 2020, pp. 1597–1607.

[31] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[32] T. Zhao, Q. Cao, and Q. Sun, "An improved approach to traceability recovery based on word embeddings," in *Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 81–89.

[33] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 133–142.

[34] A. Mahmoud, N. Niu, and S. Xu, "A semantic relatedness approach for traceability link recovery," in *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, 2012, pp. 183–192.

[35] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia, "Using code ownership to improve ir-based traceability link recovery," in *Proceedings of the 21th International Conference on Program Comprehension (ICPC)*. IEEE, 2013, pp. 123–132.

[36] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk, "Enhancing software traceability by automatically expanding corpora with relevant documentation," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2013, pp. 320–329.

[37] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang, “Improving trace accuracy through data-driven configuration and composition of tracing features,” in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (FSE)*, 2013, pp. 378–388.

[38] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, “Applying a smoothing filter to improve ir-based traceability recovery processes: An empirical investigation,” *Information and Software Technology*, vol. 55, no. 4, pp. 741–754, 2013.

[39] A. Mahmoud and N. Niu, “On the role of semantics in automated requirements tracing,” *Requirements Engineering*, vol. 20, no. 3, pp. 281–300, 2015.

[40] K. Moran, D. N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, and J. Johnson, “Improving the effectiveness of traceability link recovery using hierarchical bayesian networks,” in *Proceedings of the 42th ACM/IEEE International Conference on Software Engineering (ICSE)*, 2020, pp. 873–885.

[41] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[42] J. Mueller and A. Thyagarajan, “Siamese recurrent architectures for learning sentence similarity,” in *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, vol. 30, no. 1, 2016.

[43] “itertools — functions creating iterators for efficient looping,” https://docs.python.org/3/library/itertools.html, 2022.

[44] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2. IEEE, 2006, pp. 1735–1742.

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *ArXiv Preprint ArXiv:1810.04805*, 2018.

[46] “Coest — center of excellence for software & systems traceability,” http://coest.org, 2022.

[47] Y. Wan, W. Zhao, H. Zhang, Y. Sui, G. Xu, and H. Jin, “What do they capture?–a structural analysis of pre-trained language models for source code,” in *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022.

[48] U. Alon, S. Brody, O. Levy, and E. Yahav, “code2seq: Generating sequences from structured representations of code,” *arXiv preprint arXiv:1808.01400*, 2018.

[49] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: Learning distributed representations of code,” in *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL. ACM New York, NY, USA, 2019, pp. 1–29.

[50] Y. Sui, X. Cheng, G. Zhang, and H. Wang, “Flow2vec: value-flow-based precise code embedding,” in *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA. ACM New York, NY, USA, 2020, pp. 1–27.

[51] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 9729–9738.

[52] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, “Unsupervised learning of visual features by contrasting cluster assignments,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9912–9924, 2020.

[53] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, “Supervised contrastive learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 661–18 673, 2020.

[54] N. D. Bui, Y. Yu, and L. Jiang, “Self-supervised contrastive learning for code retrieval and summarization via semantic-preserving transformations,” in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2021, pp. 511–521.

[55] X. Cheng, G. Zhang, H. Wang, and Y. Sui, “Path-sensitive code embedding via contrastive learning for software vulnerability detection,” in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2022, pp. 519–531.