EALink: An Efficient and Accurate Pre-trained Framework for Issue-Commit Link Recovery

Chenyuan Zhang[1], Yanlin Wang[2], Zhao Wei[3], Yong Xu[3], Juhong Wang[3], Hui Li[1*] and Rongrong Ji[1]
[1]*Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China*
School of Informatics, Xiamen University, China
[2]*School of Software Engineering, Sun Yat-sen University, China*
[3]*Tencent, China*
zhangchenyuan@stu.xmu.edu.cn, wangylin36@mail.sysu.edu.cn,
{zachwei, rogerxu, julietwang}@tencent.com, {hui, rrji}@xmu.edu.cn

***Abstract*—Issue-commit links, as a type of software traceability links, play a vital role in various software development and maintenance tasks. However, they are typically deficient, as developers often forget or fail to create tags when making commits. Existing studies have deployed deep learning techniques, including pre-trained models, to improve automatic issue-commit link recovery. Despite their promising performance, we argue that previous approaches have four main problems, hindering them from recovering links in large software projects. To overcome these problems, we propose an efficient and accurate pre-trained framework called EALink for issue-commit link recovery. EALink requires much fewer model parameters than existing pre-trained methods, bringing efficient training and recovery. Moreover, we design various techniques to improve the recovery accuracy of EALink. We construct a large-scale dataset and conduct extensive experiments to demonstrate the power of EALink. Results show that EALink outperforms the state-of-the-art methods by a large margin (15.23%-408.65%) on various evaluation metrics. Meanwhile, its training and inference overhead is orders of magnitude lower than existing methods. We provide our implementation and data at https://github.com/KDEGroup/EALink.**

***Index Terms*—issue-commit link recovery, software traceability**

I. Introduction

Software traceability links maintain associations between diverse artifacts (e.g., requirements, design and source code) [1] and support various software development and maintenance tasks, e.g., software change impact analysis [2], identifying vulnerability-fixing commits [3], selective regression testing [4] and project management [5]. Unfortunately, the maintenance of software traceability links typically relies on the labor-intensive manual work of developers, leading to incomplete, inaccurate and even conflicting traceability links [6]. Hence, the deficient software traceability links increase the cost of software development and maintenance.

In this paper, we focus on recovering software traceability links among related issues and commits in software repositories, i.e., *issue-commit link recovery*. Issues summarize user's discussions around required changes of the software in the form of documentation, while commits contain the change itself with a commit message using natural language text [7]. With the ubiquitous adoption of version control systems and

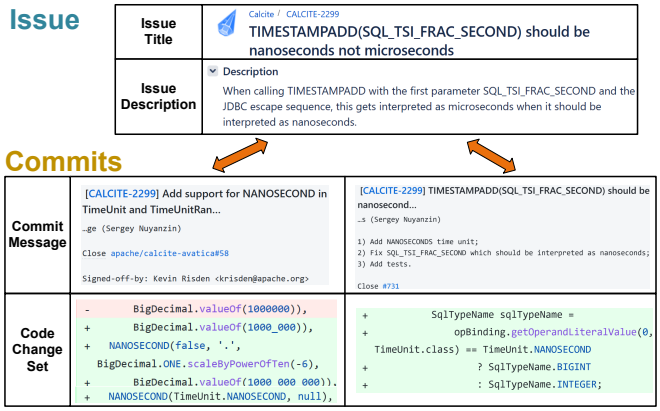

Fig. 1: An example of an one-to-many issue-commit link.

platforms such as Git and GitHub, and issue tracking systems such as Jira and Bugzilla, developers can tag commits with corresponding issues as they perform daily software development and maintenance tasks [8]. An issue is commonly fixed by one or more commits [9]. Fig. 1 shows the data format of an issue-commit link with two commits are linked to the issue via the issue tag [CALCITE-2299], we explain details of the data collection process in Sec. III. The traceability links between issues and commits play a vital role in software development and maintenance activities (e.g., commit analysis [10] and bug prediction [11]). Nevertheless, issue-commit links suffer from the same issue as other software traceability links: links are typically deficient as developers often forget or fail to create tags when making commits [12].

To alleviate the scarcity of issue-commit links, various automatic issue-commit link recovery methods are proposed. Early approaches adopt feature-based and rule-based methods [11], [13], [14] which heavily rely on feature engineering and manual rules. They show low precision and are hard to generalize. Traditional learning based approaches [7]–[9], [15], [16] leverage machine learning to automatically learn from features and alleviate the reliance on manual rules. More recently, researchers are inspired by the success of deep learning in various applications and adopt deep neural networks to enhance issue-commit link recovery [10], [17],

*Corresponding author.

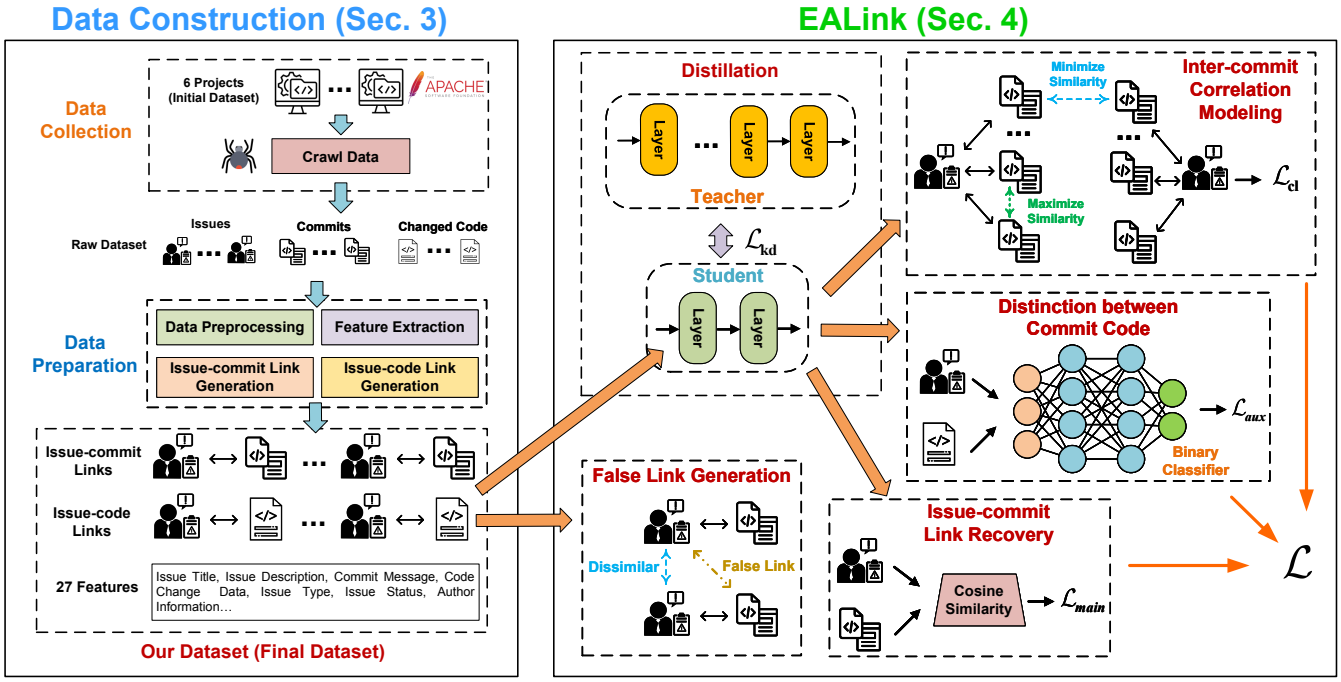

Fig. 2: Overview of this work. Orange arrows indicate the relationship among different parts in EALink.

[18]. Deep learning methods encode code and textual data into separate spaces for modeling programming language (PL) and natural language (NL) in order to overcome the semantic gap [19], [20] between different software artifacts. Moreover, they can leverage pre-training techniques and pre-train the model on related software engineering tasks and code corpora so that the issue-commit link recovery task can benefit from large-scale code data and supervision beyond issue-commit link data [18].

Although much effort has been devoted to interlinking issues and commits, we find that there still exist problems:

1) **P1: High Training and Inference Overhead:** Despite the success of deploying pre-trained models (e.g., BERT [21]) in issue-commit recovery [18], the performance improvement comes at the cost of much more model parameters and longer training and inference time, hindering their use in large-scale software projects. For example, T-BERT [18] takes 44,353 seconds for 1,000 recovery queries on a moderate-size dataset in our experiments.
2) **P2: Neglect of Inter-commit Correlation:** In practice, multiple commits may correspond to the same issue. Prior methods mostly model links between an issue and each corresponding commit, neglecting correlations among commits in a one-to-many issue-commit link (i.e., inter-commit correlation).
3) **P3: Nondistinctive Modeling of Changed Code in One Commit:** Existing approaches treat all changed code files in a commit equally. However, this is not reasonable in practice. Analysis on code repositories demonstrates that changes for different purposes can be submitted together in a single commit [16]. Inappropriately modeling loosely related and unrelated code changes introduces noise, jeopardizing learning quality.
4) **P4: Conflicting False Links:** Many deep learning based recovery approaches are trained via separating true links and false links. True links that correctly connect issues to their related commits can be extracted from tags provided on GitHub, Jira and Bugzilla. To construct inexistent false links that interlink issues and unrelated commits, existing methods adopt a sampling method based on the time interval. However, such a method may generate false links that are actually true links. See Sec. II for details.

We provide motivating examples for each problem in Sec. II. To tackle the above problems, we propose an Efficient and Accurate Pre-trained framework for issue-commit Link recovery (EALink). Fig. 2 provides an overview of this work and our contributions are:

- To reduce the overhead brought by large model size (**P1**), EALink distills knowledge from a pre-trained NL-PL model (e.g., CodeBERT [22]) to construct a compact model. The distilled model can capture the semantic connections between NL and PL, which is essential to model issues and commits. Meanwhile, it is easier to fine-tune the compact model on the issue-link recovery task since it contains fewer parameters and requires much shorter training and inference time.
- To model inter-commit correlation (**P2**), EALink employs contrastive learning [23]. And inter-commit correlation is captured by contrasting positive commits (from the same one-to-many issue-commit link) and negative commits (from different issue-commit links).

- To provide a fine-grained distinction between data of changed code files (**P3**), we design an auxiliary task *issue-code link prediction* to help distinguish the importance of each commit code. The task is jointly trained with the issue-commit link recovery task in a manner of multi-task learning.
- To avoid conflicting false links (**P4**), we propose a false link generation mechanism for constructing reasonable false links used for training EALink.
- To fairly evaluate EALink and existing works on large-size projects, we construct a new, large dataset for the issue-commit link recovery task (it also contains issue-code links for our designed auxiliary task). Extensive experiments demonstrate that EALink recovers issue-commit links more efficiently and accurately than state-of-the-art baselines on large projects.

II. Motivating Examples

In this section, we provide detailed motivating examples to illustrate the four problems mentioned in Sec. I.

Example for P1 (High Training and Inference Overhead): Recent deep learning based issue-link recovery models show promising performance [10], [18] as they can better capture textual features, code features and their mutual relations via deep neural networks. However, the strong expressive power of deep neural networks, especially pre-trained models [21], [24], comes at the cost of requiring many more model parameters and longer training/inference time than traditional methods. For instance, deep learning based method DeepLink [10] and pre-training based method T-BERT [18] take 1,145 seconds and 44,353 seconds for inference on Isis, a moderate-size project in our constructed dataset. And T-BERT requires about 138 hours for pre-training using our hardware environment. The high training and inference cost of deep learning based recovery methods limits their use in processing large software projects.

Example for P2 (Neglect of Inter-Commit Correlation): In a software project, developers may make multiple commits for fixing an issue, forming one-to-many issue-commit links. One-to-many links are ubiquitous. Fig. 3 demonstrates the statistics of one-to-one and one-to-many links in the 6 projects in our constructed data (Sec. III). We can observe that all the 6 software projects have roughly similar numbers of one-to-one and one-to-many links. Multiple commits in a one-to-many link may be correlative. In the example of Fig. 1, the two commits together fix the issue [CALCITE-2299]. However, existing works will model the two commits separately, ignoring that they are related to each other.

Example for P3 (Nondistinctive Modeling of Changed Code in One Commit): Commits often contain changed code located in different source code files and some of them are loosely related or even unrelated to the issue. Fig. 4 depicts a commit for the issue [CALCITE-1094]. The changes of the source code file "UnsynchronizedBuffer.java" is relevant to

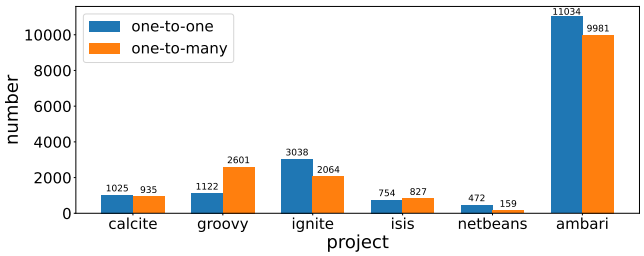

Fig. 3: Statistics of one-to-one and one-to-many links.

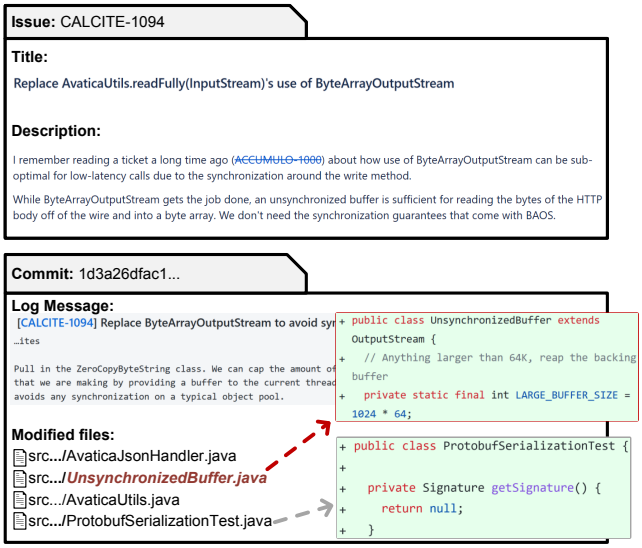

Fig. 4: Loosely related or unrelated code in a commit.

[CALCITE-1094], while the changes of "ProtobufSerializationTest.java" implement a test program and they are loosely related or unrelated to [CALCITE-1094]. Loosely related or unrelated source code files introduce noise to issue-commit link recovery. Noise hinders recovery model from capturing critical information. Most existing approaches do not take special actions to handle loosely related or unrelated source code file data. A few works [16] filter changed code that does not share many terms with issues to avoid noise. But their methods rely on keyword matching that is inappropriate for solving this problem: a) Commits (code) and issues (title and description) are in different modalities (NL and PL) that do not share the same vocabulary; b) Commits and issues may contain relevant but not exactly matching terms and using keyword matching will filter such useful, related code snippets.

Example for P4 (Conflicting False Links): Previous works generate false links by sampling commits submitted 7 days before or after one of the three dates (creation/updated/resolved dates) of an issue and connect the sampled commits to the issue as false links [10], [13], [16], [25]. However, in practice, many factors including active developers and simple issues make it possible that some commits are submitted shortly after the creation of the issue. For instance, the issue [CALCITE-

TABLE I: Statistics of our constructed final dataset.

Project	#Issues	#Commits	#True Issue-commit Links	#True Issue-code Links
Ambari	25,162	38,872	35,597	52,530
Calcite	3,740	6,934	3,058	7,910
Groovy	9,118	30,633	8,851	9,208
Ignite	12,495	32,930	9,997	17,747
Isis	2,264	15,284	8,486	27,127
Netbeans	3,705	19,181	1,369	4,662

1700] has a commit[1] submitted within 7 days after the creation of [CALCITE-1700]. Therefore, unlabeled true links may be wrongly sampled as false links using time interval based sampling method adopted by previous works.

III. Dataset Construction

Existing works adopt small datasets for evaluation. For example, Lin et al. [18] uses three software projects with hundreds of issues, commits and links in the evaluation. However, the effectiveness and efficiency of recovery models cannot be fully evaluated on small-scale data.

Hence, we constructed a large issue-commit link dataset based on 6 Java projects to evaluate link recovery methods. The data for each project contains true links ranging from one thousand three hundred to thirty five thousand. Moreover, it includes 27 non-text and text features. Tab. I depicts our constructed final dataset and Tab. II provides detailed descriptions of the 27 features. Note that not all the features are used in this study. Unused features could be helpful for future studies on issue-commit link recovery.

This section shows the steps of dataset construction which are depicted in the left of Fig. 2. For the purpose of illustration, in the following sections, *a link refers to a one-to-one issue-commit link and a one-to-many issue-commit link is broken into multiple one-to-one issue-commit links.*

A. Data Collection

Claes et al. [26] present a dataset containing common issue and commit information (i.e., issue title, issue descriptions, issue comments, commit messages and meta-information of changed source code files) on 765 projects across 20 years. However, it lacks information of code change data and source code file data in commits. We selected 6 Apache projects from their dataset based on project popularity (i.e., the number of stars and forks) as the *initial dataset*. After that, we crawled code change data and source code data based on the meta-information of changed source code files that contains urls of code data. The crawled code data was added to the initial dataset as a supplement to form the *raw dataset*.

B. Data Preparation

We prepared the *final dataset* by conducting four operations on the raw dataset: data preprocessing, feature extraction, true issue-commit link generation and issue-code link generation.

[1]https://github.com/apache/calcite/commit/8bac70e5020c50505b34df6eaae18c0ca414f2c2

TABLE II: Descriptions of 27 features in final dataset.

Feature	Description
source	Describe which platform or system the project originates from (e.g., apache).
product	Project name (e.g. Netbeans).
issue_id	ID number corresponding to an issue (e.g., 13226408).
component	The component type that the project belongs to (e.g., engine).
creator_key	A hash value that uniquely identifies the issue (e.g., 7860ba1e91f42c...).
create_date	The creation time of the issue (e.g., 2018-03-13 09:06:55+00:00).
update_date	The last update time of the issue.
last_resolved_date	The last resolved time of the issue.
comment	Discussion on the issue.
summary	Summary of the issue.
description	The specific content of the issue.
issue_type	The type of the issue (e.g., bug/improvement/new feature).
status	The status of the issue (e.g., open/close/resolved).
repo	The name of the project that the commit belongs to.
commitid	A hash value that uniquely identifies the commit.
parents	The hash value of the lasted submission.
author	Hash for the author of commit.
committer	The hash of the person who made the commit.
author_time_date	The time when the author submitted the commit.
commit_time_date	The time when the committer submitted the commit.
message	Summary of the content of the commit.
commit_issue_id	If the commit has been marked, the value of commit_issue_id is the issue id.
changed_files	The path of the changed file.
Diff	Code diff.
codelist	List of source code files.
nonsource	Content of changed non-source code files.
label	Its value is 1 if it corresponds to a true link, otherwise 0.

1) Data Preprocessing: The raw dataset contains information (e.g, issue description) created by different developers with diverse development conventions. Hence, different text styles and programming styles are manifested in the raw dataset, bringing noise. To improve data quality, we adopted several strategies to preprocess textual and code data contained in the raw dataset.

Text Preprocessing: The textual data in the raw dataset contains issue title, issue description and commit message. We first performed several common NLP preprocessing strategies including lowercase conversion, tokenization, stop word removal and stemming [27] on textual data. They not only reduced the size of the token vocabulary, thus allowing for a compact feature set, but also integrated different forms of words by replacing them with root words. Then, following DeepLink [10], we removed hyperlinks and issue tags from textual data. Hyperlinks were removed since they are typically not viewed as textual data. Issue tags were removed since

all commit messages in some project repositories[2] contain issue tags (e.g., the commit message in Fig. 1). Since issue tags can be directly used to find true links (see Sec. III-B3), we removed them to avoid data leakage and increase the difficulty of issue-commit link recovery. We also identified inline code and large code blocks from textual data using regular expressions. Identified issue code was removed from textual data but was included as features in the final dataset (see Sec. III-B2).

Code Preprocessing: We first extracted identifier names (e.g., class names, method names and variable names) from code data. Moreover, for source code data, since it contains the complete function body, we used tree-sitter[3] to convert it to Abstract Syntax Tree (AST) and extract identifier names from node tokens. For code change data, we used the patterns (i.e., regular expressions) proposed by Nguyen et al. [13] and Sun et al. [16] to extract identifier names. Then, the extracted identifier names were split into tokens according to their patterns. Finally, all code data was converted to lowercase.

2) Feature Extraction: The features extracted and used by EALink are *issue title*, *issue description*, *commit message*, *code change* and *changed source code files in each commit*. Note that this paper mainly focuses on the model design for the link-commit issue recovery task and EALink only considers the above five features. We additionally extracted many other textual and non-textual features (e.g., issue status and updated date) and included them in the final dataset since some issue-commit link recovery approaches require additional features. For instance, DeepLink [10] requires *issue code* which includes code tokens in issue descriptions. KG-DeepLink [17] uses a feature vector including issue type, issue priority, issue reporter, issue assignee, issue created time, issue resolved time, commit time and committer. In total, our final dataset contains 27 features and these features can be useful to future study of link-commit issue recovery.

3) True Issue-Commit Link Generation: If a commit is marked with an issue tag, we labeled it together with the corresponding issue as a *true link* and added to the final dataset. False links were generated during training (see Sec. IV-E).

4) Issue-Code Link Generation: EALink is also trained on the auxiliary task issue-code link prediction which will be described in Sec. IV-D. To prepare the data for the auxiliary task, for each true issue-commit link, we simply checked whether the file name of each changed source code file appeared in issue title or issue description. If so, we constructed an issue-code link between the changed source code file and the corresponding issue, and added it to the final dataset.

IV. Our framework EALink

A. Overview

The right part of Fig. 2 depicts EALink which contains two steps. In the first step, EALink distills knowledge of a large teacher model to construct a compact student model (Sec. IV-B). In the second step, EALink fine-tunes the lightweight student model for the task of issue-commit link recovery. Specifically, during fine-tuning, EALink is trained to capture inter-commit correlation via contrastive learning (Sec. IV-C), and distinguish the varying importance of different commit code through the auxiliary task of issue-code link prediction (Sec. IV-D). In fine-tuning, EALink is optimized in a manner of multi-task learning and we propose a false link generation approach to prepare the links for training (Sec. IV-E).

Model Input: The input to the first step (distillation) is a code pre-trained model. The input to the second step (fine-tuning) includes *issue text* (i.e., the concatenation of the title and description of the issue), *commit message*, *commit code* (i.e., code data in one changed file of the commit) and *code change* (i.e., difference between two versions of the changed code file).

Notation: $\mathcal{B} = \{t_1, t_2, \cdots, t_{|\mathcal{B}|}\}$ indicates a batch of issue-commit links, where $t_i = \{s_i, q_i\}$ and $t_i \in \mathcal{B}$. s_i and q_i denote the issue and the commit in the one-to-one issue-commit link t_i. $\mathcal{B}$ includes both true links $\mathcal{B}_{\text{true}}$ and false links $\mathcal{B}_{\text{false}}$. The complete input data X is randomly divided into multiple batches (i.e., multiple $\mathcal{B}$) in each training iteration following the standard machine learning training paradigm. Other notations are explained when they are used.

B. Distillation of Code Pre-training Model (P1)

The software engineering community has embraced the use of pre-trained models that have achieved a great success in Natural Language Processing and Computer Vision domains [21], [28], and there is a surge of works on pre-trained models [22], [29], [30] for code-related tasks recently. For the issue-commit link recovery task, pre-training has been firstly adopted in T-BERT [18].

The effectiveness of pre-trained models can be attributed to their ability to learn universal representations from a massive amount of mostly unlabeled data through self-supervised learning tasks to benefit various downstream tasks with relatively much less data. This leads to a better starting point for model optimization, improved generalization performance, and a form of regularization to prevent overfitting on small datasets [31]. Despite the empirical success, a widely recognized problem of pre-trained models is the computational efficiency problem, as they often have a large number of parameters and take a long time for training and inference [32], [33].

EALink is also designed based on pre-training and it can adopt any code pre-trained model to have all the benefits of pre-training. To reduce the size of the pre-trained model as well as the training and inference time, we adopt the idea of knowledge distillation [34] to transfer the knowledge encoded in the large pre-trained model (i.e., teacher) to a compact student model. The student model is later used for issue-commit link recovery in EALink. Fig. 5 illustrates the distillation process.

[2]The Calcite repository is one example: https://github.com/apache/calcite.
[3]https://github.com/tree-sitter/tree-sitter

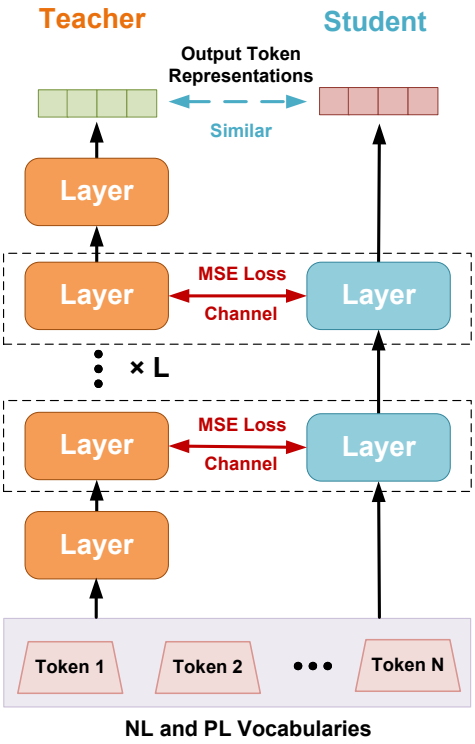

Fig. 5: Distillation for the code pre-trained model.

Teacher Model: Let $f_t(X;\theta)$ denote the code pre-trained teacher model where X is the input to EALink and θ is model parameters. Prevalent code pre-trained models typically adopt a BERT-style architecture which contains a multi-layer bidirectional Transformer [35] with L Transformer blocks. For instance, CodeBERT [22] uses exactly the same architecture as RoBERTa-base [24] with 12 layers. Since the use of Transformer has become ubiquitous, we omit the background description of Transformer.

Student Model: The distillation component aims to learn a much smaller parameter set θ' for a compact student model $f_s(X;\theta')$ which shows similar performance to the large pre-trained teacher model. In EALink, we use a two-layer RoBERTa (i.e., two Transformer blocks) as the student model.

Distillation Strategy: A common and easy-to-deploy distillation strategy is the last-layer distillation [34], i.e., the student imitates the output from the last layer of the teacher model. However, as the number of training epochs increases, the student model using last-layer distillation may face the poor generalization issue caused by overfitting on the training data [32]. Therefore, we opt to the intermediate-layer distillation strategy [36] that distills encoded knowledge from intermediate layers in the teacher model. The hidden states of all NL and PL tokens in a teacher's hidden layer are used as hints to train the hidden states in a student's layer so that the hidden states of corresponding teacher and student layers are close. To avoid high overhead, we train each of the two layers in the student model to mimic one intermediate layer in the teacher model and each teacher-student distillation layer pair is called as a distillation *channel* as shown in Fig. 5. We use a dimension-wise MSE loss (i.e., squared Euclidean distance) in distillation:

$$\mathcal{L}_{\text{kd}} = \sum_{\langle l_t, l_s\rangle \in C} \sum_{v \in \mathcal{V}} \sum_{z=1}^{d} \left(\mathbf{h}_{v,z}^{(l_t)} - \mathbf{h}_{v,z}^{(l_s)}\right)^2, \tag{1}$$

where C denotes all pre-set distillation channels, $\mathcal{V}$ is the vocabulary containing all pre-trained NL and PL tokens, and d indicates the dimensionality of hidden states. $\mathbf{h}_{v,z}^{(l_t)}$ and $\mathbf{h}_{v,z}^{(l_s)}$ indicates the z-th dimension of the hidden state for the token v output in the l_t-th layer of the teacher model and the l_s-th layer of the student model, respectively. After the distillation, the student model can produce representations for NL and PL tokens that are similar to the teacher model's output representations. And the student model has much fewer parameters, making it more easy to adopt the student model in fine-tuning and inference.

C. Inter-Commit Correlation Modeling (P2)

As discussed in previous sections, multiple commits may correspond to the same issue and we believe this provides additional supervision to help better model commits. Since commits belonging to the same one-to-many issue-commit link should be related to each other (e.g., they together fix a bug), we design an inter-commit correlation modeling component in EALink to capture such correlations.

Specifically, this component is designed with the idea of contrastive learning [23]. Contrastive learning has become prevalent representation learning paradigm as it does not require labels and can learn such a representation space where similar sample pairs stay close to each other while dissimilar ones are far apart. For a commit q_i in an issue-commit link $t_i = \{s_i, q_i\}$ from a batch of true issue-commit links $\mathcal{B}_{\text{true}}$, we sample another link $t_j = \{s_j, q_j\}$ from $\mathcal{B}_{\text{true}}$ and use the commit q_j as the positive sample of q_i. t_i and t_j belong to the same one-to-many link, i.e., s_i and s_j are identical. If no such q_j exists, q_i is treated as the positive sample of itself. We do not sample negative samples explicitly. Instead, we treat other $|\mathcal{B}_{\text{true}}|-2$ links within $\mathcal{B}_{\text{true}}$ (exclude all other links in the same one-to-many link as t_i and t_j) as negative examples.

The representation $\mathbf{q}_i$ of a commit q_i is obtained via: $\mathbf{q}_i = \mathbf{m}_i \oplus \mathbf{c}_i$, where $\oplus$ is the concatenation operation. $\mathbf{m}_i$ and $\mathbf{c}_i$ are representations of commit message and commit code generated by the student model, respectively. For contrastive instance discrimination, we adopt a non-parametric contrastive learning loss [37] by using cosine similarity as similarity measure:

$$\mathcal{L}_{\text{cl}}(\mathcal{B}_{\text{true}}) = -\sum_{t_i \in \mathcal{B}_{\text{true}}} \log \frac{e^{\text{sim}(\mathbf{q}_i, \mathbf{q}_i^+)/\tau}}{\sum_{j \in \mathcal{N}(t_i)} e^{\text{sim}(\mathbf{q}_i, \mathbf{q}_j)/\tau}}, \tag{2}$$

where $\mathcal{L}_{\text{cl}}(\mathcal{B}_{\text{true}})$ is the loss for true links $\mathcal{B}_{\text{true}}$ in a batch $\mathcal{B}$, $\mathcal{N}(t_i)$ is the in-batch negative samples of the link t_i, $\mathbf{q}_i^+$ denotes the representation of the positive sample of the commit q_i, and $\text{sim}(\cdot)$ indicates the inner product. τ is a temperature

hyperparameter that controls the concentration level of the distribution. Summing up $\mathcal{L}_{\text{cl}}(\mathcal{B}_{\text{true}})$ for all batches, we have the complete contrastive learning loss $\mathcal{L}_{\text{cl}}$.

D. Distinction between Commit Code (P3)

To eliminate the noise brought by loosely related and unrelated commit code, we define an auxiliary task, issue-code link prediction to enhance the understanding of EALink on the importance of each commit code. The issue-code link prediction task estimates the relevance between issues and commit code.

To be specific, each issue-code link $p_i = \langle s_i, c_i \rangle$, where c_i indicates the first k_{PL} code tokens in the code change of a changed code file, is fed to the student model. $\mathbf{s}_i$ and $\mathbf{c}_i$ are representations of the issue text s_i and the code c_i in the link p_i, respectively. They are the concatenation of their tokens' representations. Similar to commit code, we preserve the first k_{NL} tokens in each issue text. Then, $\mathbf{s}_i$ and $\mathbf{c}_i$ are passed to a binary classifier which is a two-layer feedforward neural network (FFN):

$$\begin{aligned} \mathbf{s}'_i = \text{AVG}(\mathbf{s}_i), \;\; \mathbf{c}'_i = \text{AVG}(\mathbf{c}_i) \\ \hat{\mathbf{y}}_{\langle s_i, c_i \rangle} = \text{f}_2\Big(\text{f}_1\big(\mathbf{s}'_i \oplus \mathbf{c}'_i \oplus |\mathbf{c}'_i - \mathbf{s}'_i|\big)\Big), \end{aligned} \tag{3}$$

where $\text{AVG}(\cdot)$ indicates average pooling, $\text{f}_1(\cdot)$ is a single-layer FFN with the Tanh activation, and $\text{f}_2(\cdot)$ is a single-layer FFN without activation. $\hat{\mathbf{y}}_{\langle s_i, c_i \rangle} \in \mathbb{R}^2$ indicates the probabilities of the existence and the non-existence of the issue-code link p_i.

Based on the data we construct in Sec. III-B4 for issue-code link prediction and the estimated probabilities $\mathbf{y}$ of the issue-code link, we train EALink with an auxiliary objective:

$$\mathcal{L}_{\text{aux}} = -\sum_{x \in X_{\text{aux}}} \sum_{\langle s,\, c \rangle \in x} y_{\langle s,\, c \rangle} \cdot \log \hat{\mathbf{y}}_{\langle s,\, c \rangle}, \tag{4}$$

where X_{aux} is the true issue-commit link set in the auxiliary dataset, x is a true issue-commit link in X_{aux}, s indicates the issue text in x, c is the commit code in one changed file of a commit in x, and $y_{\langle s,\, c \rangle}$ is the binary label for the relevance between s and c.

We train the issue-code link prediction task together with the issue-commit link recovery task in a manner of multi-task learning [38] (illustrate in Sec. IV-E). Multi-task learning helps EALink learn from related tasks and it enhances the understanding of EALink on the importance of each commit code, avoiding the negative impact of loosely related and unrelated commit code. Unlike FRLink [16] that directly filters commit code sharing few similar terms with issue text, our approach is learning based and does not filter any commit code. This way, we do not need to manually set a similarity threshold for filtering commit code and avoid the case that the commit code is semantically related but it shares few similar terms with the issue text. Moreover, loosely related and unrelated commit code is still leveraged in training: they help train EALink to better distinguish commit code.

E. Putting All Together

The main task of EALink (i.e., the issue-commit link recovery task) is optimized via the following objective used in existing works [10]:

$$\mathcal{L}_{\text{main}} = \sum_{x \in X} \sum_{\langle s,\, q \rangle \in x} \big| y_{\langle s,q \rangle} - sim(\mathbf{s}, \mathbf{q}) \big|, \tag{5}$$

where X is the issue-commit link set, x is a link, s is the issue in x, q is a commit in x, $y_{\langle s,q \rangle}$ is the binary label (1 if $\langle s, q \rangle$ is a true link, otherwise 0), and $sim(\cdot)$ is the cosine similarity. $\mathbf{s}$ is the representation of s output by the student model and $\mathbf{q}$ is the representation of q obtained similar as in Sec. IV-C.

False Link Generation (P4): X includes both true links (extracted by tags) and false links. The main issue of existing false link generation methods [10], [13], [16], [25] is that they require a time threshold (e.g., 7 days) to filter commits. But it is hard to set a reasonable time threshold. To overcome this problem, we opt to design a similarity based false link generation method. To be specific, for the issue s in a known true link, EALink finds its least similar issue s' by comparing cosine similarity between representations of their issue text. We connect the commit q, which is originally linked to s in a true link, to s' in order to construct a false link. If multiple possible q exists (i.e., one-to-many issue-commit link), we pick the first commit scanned by EALink. The main overhead of the above method is caused by calculating all-pair cosine similarities and the time complexity is $O(|\mathcal{T}|^2)$ where $|\mathcal{T}|$ is the number of all issues. To reduce the cost of generating false links, we only search irrelevant issues contained in the same batch of true links ($\mathcal{B}_{\text{true}}$) instead of all issues during training and the time complexity reduces to $O(|\mathcal{B}_{\text{true}}|^2)$.

As illustrated in Sec. IV-D, EALink is optimized in a manner of multi-task learning and the complete objective is as follows:

$$\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda_{\text{cl}} \cdot \mathcal{L}_{\text{cl}} + \lambda_{\text{aux}} \cdot \mathcal{L}_{\text{aux}}, \tag{6}$$

where λ_{cl} and λ_{aux} are hyper-parameters for task weights. Through multi-task learning, EALink is fine-tuned to leverage useful information contained in multiple related tasks that may be difficult to learn in the main task to improve the recovery performance.

When making predictions, EALink calculates the cosine similarity between the representations $\mathbf{s}$ and $\mathbf{q}$ of the target issue s and the target commit q. $\mathbf{s}$ and $\mathbf{q}$ are obtained in a similar way as in Eq. 5. Following FRLink [16] and DeepLink [17], the similarity threshold is set to 0.5, which means that if the similarity is larger than 0.5, EALink will predict $\langle s, q \rangle$ as a true link.

V. Experiments

A. Evaluation Settings

We choose four prevalent metrics Precision@k (P@k), Normalized Discounted Cumulative Gain (NDCG@k), Mean Reciprocal Rank (MRR) and Hit Ratio (Hit@k) for evaluation.

TABLE III: Comparisons between different methods. Best performance on a project w.r.t. a metric is shown in bold.

	EALink						T-BERT					
Project	**P@1 (Hit@1)**	**P@10**	**Hit@10**	**MRR**	**NDCG@1**	**NDCG@10**	**P@1 (Hit@1)**	**P@10**	**Hit@10**	**MRR**	**NDCG@1**	**NDCG@10**
Ambari	**0.9490**	**0.1231**	**0.9800**	**0.9622**	**0.5988**	**0.6130**	0.5524	0.0775	0.7748	0.6321	0.3485	0.4405
Calcite	**0.6555**	**0.0920**	**0.8566**	**0.7313**	**0.4136**	**0.4955**	0.5540	0.0836	0.8357	0.6450	0.3495	0.4626
Groovy	**0.6650**	**0.1029**	**0.8310**	**0.7323**	**0.4196**	**0.4919**	0.4655	0.0718	0.7175	0.5482	0.2937	0.3937
Ignite	0.3950	**0.0907**	**0.7210**	0.5054	0.2492	0.3814	**0.4304**	0.0709	0.7087	**0.5273**	**0.2716**	**0.3843**
Isis	0.2423	**0.0983**	**0.5184**	0.3397	0.1529	0.2622	0.2394	0.0443	0.4304	0.3049	0.1510	0.2221
Netbeans	**0.3273**	**0.0824**	**0.6727**	**0.4567**	**0.2065**	**0.3519**	0.2794	0.0507	0.5074	0.3614	0.1763	0.2638
Average	**0.5390**	**0.0982**	**0.7633**	**0.6213**	**0.3401**	**0.4327**	0.4202	0.0665	0.6624	0.5032	0.2651	0.3612
Improve	-	-	-	-	-	-	**(↑28.27%)**	**(↑47.67%)**	**(↑15.23%)**	**(↑23.47%)**	**(↑28.29%)**	**(↑19.80%)**
	DeepLink						**VSM**					
Project	**P@1 (Hit@1)**	**P@10**	**Hit@10**	**MRR**	**NDCG@1**	**NDCG@10**	**P@1 (Hit@1)**	**P@10**	**Hit@10**	**MRR**	**NDCG@1**	**NDCG@10**
Ambari	0.0625	0.0238	0.2167	0.1266	0.0394	0.0992	0.4209	0.0680	0.6798	0.5103	0.2655	0.3698
Calcite	0.0926	0.0185	0.1739	0.1346	0.0584	0.0909	0.2770	0.0676	0.7512	0.4919	0.1866	0.3677
Groovy	0.0062	0.0176	0.1522	0.0587	0.0039	0.0554	0.4983	0.0776	0.7763	0.5936	0.3144	0.4284
Ignite	0.0584	0.0182	0.1700	0.1100	0.0368	0.0813	0.1012	0.0581	0.5806	0.3256	0.1269	0.2765
Isis	**0.3291**	0.0538	0.4960	**0.3994**	**0.2076**	**0.2787**	0.1191	0.0471	0.4705	0.2264	0.0751	0.2065
Netbeans	0.0870	0.0224	0.1925	0.1293	0.0549	0.0917	0.0125	0.0375	0.3750	0.2040	0.0789	0.1710
Average	0.1060	0.0257	0.2336	0.1598	0.0668	0.1162	0.2382	0.0593	0.6056	0.3920	0.1746	0.3033
Improve	**(↑408.65%)**	**(↑281.85%)**	**(↑226.83%)**	**(↑288.88%)**	**(↑408.88%)**	**(↑272.38%)**	**(↑126.28%)**	**(↑65.60%)**	**(↑26.04%)**	**(↑59.49%)**	**(↑94.79%)**	**(↑42.66%)**

In our evaluation, a query refers to a true link and its top-k ranking list consists of k ranked candidate commits.

- **Precision@k** indicates the percentage of relevant commits to the query in the top-K ranking list predicted by a model:

$$\text{Precision@}k = \frac{1}{|\mathcal{Q}|}\sum_{i\in\mathcal{Q}}\frac{\text{Rel}_i}{\text{k}}, \tag{7}$$

  where $\mathcal{Q}$ is the query set, $|\mathcal{Q}|$ is the number of queries, Rel_i represents the number of relevant commits that appear in the top-k ranking list for the i-th query.
- **NDCG@k** evaluates the ranking position of relevant commits:

$$\text{NDCG@}k = \frac{1}{Z_k}\sum_{i=1}^{k}\frac{2^{r_i}-1}{\log_2(i+1)}, \tag{8}$$

  where Z_k is a normalizer which ensures that perfect ranking has a value of 1, r_i is the relevance of commit at position i. We use simple binary relevance: $r_i = 1$ if the commit is actually linked to the target issue, and 0 otherwise.
- **MRR** measures whether the relevant commits to an issue are placed in more prominent positions in the ranking list:

$$\text{MRR} = \frac{1}{|\mathcal{Q}|}\sum_{|i=1|}^{|\mathcal{Q}|}\frac{1}{\text{Rank}_i}, \tag{9}$$

  where Rank_i refers to the ranking of the relevant commit.
- **Hit@k:** refers to the probability that a relevant commit is in the predicted top-k ranking list:

$$\text{Hit@}k = \frac{1}{|\mathcal{Q}|}\sum_{i}^{|\mathcal{Q}|}\mathbb{I}\left(\text{Rank}_i \le k\right) \tag{10}$$

  where $\mathbb{I}(\cdot)$ is an indicator function. If the condition is true, it returns 1, otherwise 0. Hit@1 is identical to P@1.

For each project, true links were randomly divided by 6:2:2 for training, validation and test. To accelerate the evaluation, we randomly sampled 1,000 unique issues at most from each project's test set. If the number of unique issues was less than 1,000, we selected all the unique issues. In each test set, true links $\mathcal{E}$ containing the selected issues were picked. For each true link $t_i = \{s_i, q_i\}$ in $\mathcal{E}$, we randomly sampled 99 other true links in $\mathcal{E}$. Let $t_j = \{s_j, q_j\}$ be one of the 99 sampled links. s_i and s_j should not be identical, i.e., t_i and t_j do not belong to the same one-to-many issue-commit link. Then, we connected q_j to s_i to construct a false link for t_i. For each true link in $\mathcal{E}$, we constructed 99 false links. In the experiments, we evaluate whether the model can rank a true link ahead of its corresponding 99 false links.

B. Environment and Hyper-parameter Settings

We implemented EALink using PyTorch. The experiments were run on a machine with two Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz, 256 GB main memory and one NVIDIA GeForce RTX 3090.

The default teacher model in EALink is CodeBERT [22] and we used hyper-parameters provided by the authors. We also explored other code pre-trained models in our experiments. The student model had 2 layers, 1 head and 768 hidden dimensions. By default, we set distillation channels as follows: the first layer of teacher was connected to the first layer of student, and the fifth layer of teacher was connected to the second layer of student. Results of other settings for distillation channels are reported in RQ4 of Sec. V-D. λ_{cl} and λ_{aux} were set to 1 by default. The length of tokens in each natural language text (e.g., k_{NL} in Sec. IV-D) and the length of tokens in each code data (e.g., k_{PL} in Sec. IV-D) were set to 35 and 80, respectively. To train the model, we employed the Adam optimizer [39], and the batch size was set as 16. The initial learning rate was set to $4e^{-5}$ and was multiplied by 0.8 every 6 epochs. We set hyper-parameters for baselines as suggested by their authors.

C. Baselines

We compare EALink with three representative and publicly available issue-commit link recovery approaches:

- **T-BERT**[4] [18] is the state-of-the-art pre-trained method. It is first pre-trained on the large-scale CodeSearchNet [40] for the code search task [41]. Then, the pre-

[4]https://github.com/jinfenglin/TraceBERT

trained T-BERT is fine-tuned on the small issue-commit link data for the link recovery task.

- **DeepLink**[5] [10] is a deep learning based method which adopts word embedding and RNN to learn the semantic representations of issues and commits for interlinking issues and commits.
- **VSM**[6] [18], [42] is an information retrieval based method. It expresses code and textual data as bags of words which are further represented as word vectors. The relevance between an issue and a commit is calculated based on the similarity between their corresponding word vectors.

We set all the hyper-parameters for baselines as specified in the original papers.

D. Experimental Results and Analysis

Next, we report and analyze the results of our experiments in order to answer five research questions:

RQ1: Does EALink outperform state-of-art baselines?

We compare EALink with three baselines on the 6 Java projects in our dataset. Tab. III reports the performance of all methods and the improvement percentage of EALink over baselines. In Tab. III, the best performance on a project w.r.t. a metric is shown in bold.

Tab. III shows that, in a few cases, baselines outperforms EALink. However, they do not show robust performance as EALink. For example, T-BERT is 0.04 higher than EALink in one project Ignite on P@1 and DeepLink outperforms EALink in the Isis project, while on average, EALink achieves 0.5390 on P@1 (Hit@1), 0.0982 on P@10, 0.7633 on Hit@10, 0.6213 on MRR, 0.3401 on NDCG@1 and 0.4327 on NDCG@10, respectively. The lowest and highest Hit@10 are 0.5184 for the Isis project and 0.9800 for the Ambari project, respectively. The high Hit@k indicates that the relevant commit has a high probability of being ranked in the top 10 by EALink. Compared to baselines, EALink improves the quality of issue-commit link recovery by 408.65% at most, showing the effectiveness of EALink.

The pre-trained model T-BERT achieves the highest P@1, MRR, NDCG@1 and NDCG@10 on Ignite, while EALink achieves highest P@10 and Hit@10. However, on the other five projects, EALink outperforms T-BERT on all metrics. On average, EALink exceeds T-BERT by 15.23%-47.67%, showing that the superiority of EALink over the existing pre-training issue-commit recovery method. Moreover, we can conclude that the adoption of knowledge distillation to compress the size of the pre-trained model does not sacrifice the accuracy of recovery.

Surprisingly, DeepLink shows worst performance on almost all metrics and its performance is even worse than traditional information retrieval based method VSM. The exception is the Isis project, where DeepLink shows best P@1 (Hit@1), MRR, NDCG@1 and NDCG@10. DeepLink adopts deep neural networks to better capture the semantics of code and textual data. A possible reason for its subdued performance, in our opinion, is the design of the similarity module. It computes the cosine similarities between issue title vector and commit message vector, issue description vector and commit message vector, and issue code vector and commit code vector, where the term "vector" refers to the representation. Then, the maximum cosine similarity among the three calculated values is used to estimate the relevance between an issue and a commit. However, the cross-modal similarity, i.e., the similarity between the issue and the changed code in commits is not considered. Moreover, it chooses the maximum similarity value instead of considering all the three similarity values, which may cause information loss.

[5]https://github.com/ruanhang1993/DeepLink

[6]We use https://pypi.org/project/gensim to implement VSM.

TABLE IV: Total training and test time for each model on Isis. Test time is recorded for 1,000 recovery queries.

	EALink	**T-BERT**	**DeepLink**
Train (hr)	14h	138h	37h
Test (sec)	126s	44,353s	1,145s

TABLE V: Results of the ablation study on Isis.

	P@1 (Hit@1)	**MRR**	**NDCG@1**
EALink w/o aux	0.2193	0.3172	0.1384
EALink w/o cl	0.0521	0.1246	0.0329
EALink	**0.2423**	**0.3397**	**0.1529**

RQ2: Does EALink reduce the training and inference overhead compared to existing pre-trained method? (P1)

One design goal of EALink is to reduce the training and inference overhead of deep learning based link recovery method. In Tab. IV, we report the training and test time of EALink, T-BERT and DeepLink on a moderate-size project Isis among all the 6 projects in our constructed dataset. As shown in Tab. IV, DeepLink needs 1,145 seconds to evaluate 1,000 recovery queries and T-BERT needs 44,353 seconds, which indicates that the pre-trained model is heavier and more time-consuming than the simple deep learning model. EALink uses only 126 seconds, which is several orders of magnitude lower than T-BERT. Hence, EALink is more scalable to large-scale projects in the real world. In other 5 projects, similar overhead gaps can be observed. Observed from Tab. III and Tab. IV, we can conclude that the design (i.e., use a distilled, compacted student model) of EALink makes it possible to take the advantage of a large code pre-trained model while keeping low overhead: EALink shows similar or much higher performance compared to T-BERT, while it requires training and test time that is an order of magnitude shorter. In other words, EALink is suitable for deployment in large-scale projects.

RQ3: Does each component in EALink contribute to its performance? (P2, P3, P4)

Firstly, we report the results of an ablation study in Tab. V. We take the Isis project as an example to illustrate and similar trends can be observed in other projects. The ablation study allows us to evaluate the contribution of each components in EALink. In Tab. V, "EALink w/o aux" denotes the version that

TABLE VI: Results using different false link generation on Isis.

	P@1 (Hit@1)	MRR	NDCG@1
EALink$_f$	0.1843	0.2695	0.1163
EALink	**0.2423**	**0.3397**	**0.1529**

TABLE VII: Performance of using different channels on Isis.

t_1	t_2	P@1 (Hit@1)	MRR	NDCG@1
1	5	**0.2423**	**0.3397**	**0.1529**
3	7	0.2417	0.3092	0.1355
4	9	0.2301	0.3216	0.1452
6	12	0.2224	0.3235	0.1403

TABLE VIII: Performance of using different task weights on Isis.

λ_{cl}	λ_{aux}	P@1 (Hit@1)	MRR	NDCG@1
1	1	**0.2423**	**0.3397**	**0.1529**
0.5	2.5	0.1595	0.2596	0.1006
2	10	0.1187	0.2807	0.1190
3	15	0.2193	0.32	0.1384
4	20	0.2117	0.31	0.1335

TABLE IX: Performance of using different teachers on Isis.

Teacher	P@1 (Hit@1)	MRR	NDCG@1
CodeBERT	**0.2423**	**0.3397**	**0.1529**
GraphCodeBERT	0.2239	0.3171	0.1413
UniXcoder	0.2362	0.3393	0.1490

removes the issue-code prediction task proposed in Sec. IV-D and it treats all code changes in a commit equally. "EALink w/o cl" indicate the version that removes the inter-commit correlation modeling component illustrated in Sec. IV-C. From the results, we can observe that both "EALink w/o aux" and "EALink w/o cl" show worse performance compared to EALink: the MRR is reduced by 0.0225 and 0.2151 for Isis project, respectively.

The observation shows that, the designs of both the auxiliary task and the inter-commit correlation modeling enhance EALink. In other words, addressing P2 and P3 discussed in Sec. II helps improve the quality of generated links. Comparing "EALink w/o aux" and "EALink w/o cl", we can see that the inter-commit correlation modeling component contributes more to the overall performance than the auxiliary task.

We further investigate the impact of using different false link generation methods and report the results on Isis in Tab. VI. Similar conclusion can be drawn for other projects. In Tab. VI, EALink$_f$ indicates that we use the time interval based false link generation method used in previous link recovery methods [10], [13], [16], [25], which may generate conflicting false links as we discussed in Sec. II. We can see that EALink outperforms EALink$_f$ by a large margin since our proposed false link generation method produces high-quality false links and does not sample true links as false links.

In summary, based on the results reported in Tab. V and Tab. VI, we can conclude that all components in EALink contribute to its high performance in the issue-commit link recovery task.

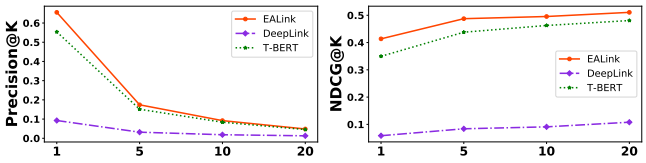

Fig. 6: Impacts of k on evaluation results. X axis shows k.

RQ4: Do different settings affect EALink?

We change several settings of EALink and investigate the impacts on EALink. We use Isis to illustrate the result and similar trends can be observed in other projects.

Firstly, we investigate the impact of changing the distillation channels of the distillation component. In Tab. VII, we report the performance when varying the channels. t_1 and t_2 indicate layers in the pre-trained teacher model that are chosen to establish the distillation channels. The t_1-th layer is connected to the first layer in the student model and the t_2-th layer is linked to the second layer in the student model. From Tab. VII, we can see that changing the setting of channels affects the performance slightly and using the default setting yields the best result.

Secondly, we investigate the impacts of changing task weights λ_{cl} and λ_{aux} in the training objective shown in Eq. 6. We report the results in Tab. VIII. When changing the values of λ_{cl} and λ_{aux}, the performance of EALink is affected. The best result is achieved using a balanced multi-task learning loss function ($\lambda_{\text{cl}} = 1$, $\lambda_{\text{aux}} = 1$). Hence, we use $\lambda_{\text{cl}} = 1$, $\lambda_{\text{aux}} = 1$ in the default setting.

Thirdly, we change the pre-trained model used as the teacher in EALink and investigate the impact. Tab. IX demonstrates the performance of using three code pre-trained models CodeBERT [22], GraphCodeBert [29] and UniXcoder [30] as the teacher in EALink on Isis. We can see that using different teachers have an impact on the performance of EALink, but the performance differences are slight, showing that EALink is flexible to accommodate different code pre-trained models. CodeBERT shows best results and we adopt it as the default teacher in EALink.

Finally, we further show the results when changing k. Fig. 6 shows the performance when using different k for evaluations. We can see that all the three methods exhibit similar trends as k changes, showing that using different k does not affect the performance rank of the three methods. In other words, using top-k ranking metrics, we can draw consistent conclusions regardless of the value of k.

In summary, changing the settings of EALink affects its performance, but most changes do not incur significant impacts.

VI. Threats to Validity

Two main threats may affect the validity of our study.

The first threat is the types and the size of the training data. Our experiments are conducted on Java projects only. We only train and evaluate recovery models on our constructed data from 6 Apache projects. The performance of EALink may vary

when applying it to other programming languages or training it on more project data. Moreover, the number of generated false links can be set to a larger number, while the number of true links is limited by the occurrence frequency of issue tags in commits. To avoid the impact of imbalanced data on training EALink, we only generate one false link for each true link in EALink.

The second threat is the reliability of true and false links. Currently, true links in our dataset are established according to the crawled issue tags. But related commits may not contain issue tags, resulting in the missing of some true links in our data. We propose a false link generation mechanism used in EALink. It connects the issue, which is least similar to the issue in a true link, to the commit in the corresponding true link for constructing a false link. Since the used issue can be completely irrelevant to the commit in the true link, the mechanism may generate false links that are too easy for EALink to identify, bringing ineffective model training.

VII. Related Work

In this section, we discuss several areas related to our work.

A. Software Traceability Recovery

Information retrieval (IR) methods are widely used in early approaches for traceability recovery. For example, several works adopt Vector Space Model [43], Latent Semantic Indexing [44], [45] and Latent Dirichlet Allocation [46], [47] to recover software traceability. However, these methods heavily rely on feature engineering and represent software artifacts as bags of words, which cannot fully capture artifacts' semantics. To solve this problem, machine learning (ML) methods are used to better capture artifacts' semantics. ENRL [48] trains seven classifiers using golden standard traceability links to detect positive traceability links in the ranking lists generated by IR methods. Similarly, TRAIL [49] trains six classifiers to automatically verify ranked links generated by IR methods. SPLINT [42] adopts semi-supervised learning to predict traceability links with unlabeled sets. To work with less training data, ALCATRAL [50] integrates active learning and supervised traceability link classifier. Comet [51] adopts probabilistic model to capture relationships between developer feedback and transitive (often implicit) relationships among groups of software artifacts.

B. Issue-Commit Link Recovery

Pioneering works on issue-commit link recovery [11], [13], [14] heavily rely on feature engineering and manual rules and are hard to generalize. Traditional learning based methods adopt machine learning (e.g., classification) to automatically learn from features. Representative approaches include but not limited to RCLinker [15], FRLink [16], PULink [9], HybridLinker [7] and the work of Rath et al. [8]. Although they overcome the drawbacks of feature based and rule based approaches to some extent, they suffer from the deficiency of shallow learning (weak express power) and lack of training data.

More recent works deploy deep learning to improve issue-commit link recovery. DeepLink [10] adopts word embedding and RNN to learn the semantic representations of issues and commits. TraceNN [20] uses similar techniques as DeepLink, but it is designed for recovering traceability links between requirements and design documents instead of issue-commit links. KG-DeepLink [17] combines RNN and SVM for predicting links based on a code knowledge graph constructed from ASTs. Beside, pre-trained models like BERT [21] also shed some light on overcoming the lack of issue and commit training data. Lüders et al. [52] adopt BERT to predict the types of links (e.g., duplicate and clone) in issue trackers. T-BERT [18] is first pre-trained on CodeSearchNet [40]. Then, it is fine-tuned on small issue-commit link data for the link recovery task. Although deep learning based methods show promising results, they do not fully address the problems discussed in Sec. II.

C. Pre-training for Code Representation Learning

There are various code-related tasks (e.g., code completion [53], code refactoring [54] and code summarization [55]) that require code representation learning. The success of BERT [21] has inspired the research on code pre-trained models that significantly benefit code-related tasks. CodeBERT [22] is a bimodal (NL and PL) pre-trained model. GraphCodeBERT [29] uses data flow to learn more comprehensive representations. UniXcoder [30] leverages cross-modal contents and mask attention matrices with prefix adapters to improve code pre-training. Unlike previous code pre-trained models that only focus on the encoder, Mastropaolo et al. [56] and Niu et al. [57] explore the seq2seq architecture in code pre-training.

VIII. Conclusion

Issue-commit links play a vital role in various software development and maintenance tasks. However, they are commonly deficient in software development and maintenance. Therefore, automatic issue-commit link recovery methods, which can reduce the cost of manual labeling and assist with various software engineering tasks, have attracted significant attention. In this paper, we point out the problems of existing issue-commit link recovery methods and propose EALink for efficiently and accurately recovering issue-commit links. EALink requires much fewer model parameters but shows better recovery results compared to existing recovery models. In the future, we will explore using other software artifacts (e.g., requirements, designs and test cases) to further improve EALink. We will also deploy other model compression techniques to further reduce the model size of EALink and expedite the recovering process.

Acknowledgment

This work was partially supported by National Key R&D Program of China (No. 2022ZD0118201), National Natural Science Foundation of China (No. 62002303, 42171456), Natural Science Foundation of Fujian Province of China (No. 2020J05001) and CCF-Tencent Open Fund.

REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Software Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[2] T. W. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in *ICPC*, 2020, pp. 14–24.

[3] G. Nguyen-Truong, H. J. Kang, D. Lo, A. Sharma, A. E. Santosa, A. Sharma, and M. Y. Ang, "HERMES: using commit-issue linking to detect vulnerability-fixing commits," in *SANER*, 2022, pp. 51–62.

[4] L. Naslavsky and D. J. Richardson, "Using traceability to support model-based regression testing," in *ASE*, 2007, pp. 567–570.

[5] M. C. Panis, "Successful deployment of requirements traceability in a commercial engineering organization...really," in *RE*, 2010, pp. 303–307.

[6] A. D. Rodriguez, J. Cleland-Huang, and D. Falessi, "Leveraging intermediate artifacts to improve automated trace link retrieval," in *ICSME*, 2021, pp. 81–92.

[7] P. R. Mazrae, M. Izadi, and A. Heydarnoori, "Automated recovery of issue-commit links leveraging both textual and non-textual data," in *ICSME*, 2021, pp. 263–273.

[8] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *ICSE*, 2018, pp. 834–845.

[9] Y. Sun, C. Chen, Q. Wang, and B. W. Boehm, "Improving missing issue-commit link recovery using positive and unlabeled data," in *ASE*, 2017, pp. 147–152.

[10] H. Ruan, B. Chen, X. Peng, and W. Zhao, "Deeplink: Recovering issue-commit links based on deep learning," *J. Syst. Softw.*, vol. 158, 2019.

[11] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: recovering links between bugs and changes," in *SIGSOFT FSE*, 2011, pp. 15–25.

[12] M. Izadi, P. R. Mazrae, T. Mens, and A. van Deursen, "Linkformer: Automatic contextualised link recovery of software artifacts in both project-based and transfer learning settings," *arXiv Preprint*, 2022. [Online]. Available: https://arxiv.org/abs/2211.00381

[13] 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 *SIGSOFT FSE*, no. 63, 2012, pp. 1–11.

[14] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. C. Gall, "Discovering loners and phantoms in commit and issue data," in *ICPC*, 2015, pp. 4–14.

[15] T. B. Le, M. L. Vásquez, D. Lo, and D. Poshyvanyk, "Rclinker: automated linking of issue reports and commits leveraging rich contextual information," in *ICPC*, 2015, pp. 36–47.

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

[17] R. Xie, L. Chen, W. Ye, Z. Li, T. Hu, D. Du, and S. Zhang, "Deeplink: A code knowledge graph based deep learning approach for issue-commit link recovery," in *SANER*, 2019, pp. 434–444.

[18] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained BERT models," in *ICSE*, 2021, pp. 324–335.

[19] A. D. Lucia, F. Fasano, and R. Oliveto, "Traceability management for impact analysis," in *2008 Frontiers of Software Maintenance*, 2008, pp. 21–30.

[20] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *ICSE*, 2017, pp. 3–14.

[21] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*, 2019, pp. 4171–4186.

[22] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," in *EMNLP (Findings)*, 2020, pp. 1536–1547.

[23] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 857–876, 2023.

[24] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *arXiv Preprint*, 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[25] C. Bird, A. Bachmann, F. Rahman, and A. Bernstein, "LINKSTER: enabling efficient manual inspection and annotation of mined data," in *SIGSOFT FSE*, 2010, pp. 369–370.

[26] M. Claes and M. V. Mäntylä, "20-mad: 20 years of issues and commits of mozilla and apache development," in *MSR*, 2020, pp. 503–507.

[27] D. D. Palmer, "Text preprocessing," in *Handbook of Natural Language Processing*. Chapman and Hall/CRC, 2010, pp. 9–30.

[28] F. Chen, D. Zhang, M. Han, X. Chen, J. Shi, S. Xu, and B. Xu, "VLP: A survey on vision-language pre-training," *Int. J. Autom. Comput.*, vol. 20, no. 1, pp. 38–56, 2023.

[29] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. B. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," in *ICLR*, 2021. [Online]. Available: https://openreview.net/pdf?id=jLoC4ez43PZ

[30] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," in *ACL (1)*, 2022, pp. 7212–7225.

[31] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, 2020.

[32] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient knowledge distillation for BERT model compression," in *EMNLP/IJCNLP (1)*, 2019, pp. 4322–4331.

[33] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling BERT for natural language understanding," in *EMNLP (Findings)*, vol. EMNLP, 2020, pp. 4163–4174.

[34] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, 2021.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[36] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *ICLR (Poster)*, 2015. [Online]. Available: https://openreview.net/forum?id=SWN-Izy8Sl9

[37] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *CVPR*, 2018, pp. 3733–3742.

[38] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 12, pp. 5586–5609, 2022.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[40] H. Husain, H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv Preprint*, 2019. [Online]. Available: https://arxiv.org/abs/1909.09436

[41] C. Liu, X. Xia, D. Lo, C. Gao, X. Yang, and J. C. Grundy, "Opportunities and challenges in code search tools," *ACM Comput. Surv.*, vol. 54, no. 9, pp. 196:1–196:40, 2022.

[42] L. Dong, H. Zhang, W. Liu, Z. Weng, and H. Kuang, "Semi-supervised pre-processing for learning-based traceability framework on real-world software projects," in *ESEC/SIGSOFT FSE*, 2022, pp. 570–582.

[43] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Software Eng.*, vol. 32, no. 1, pp. 4–19, 2006.

[44] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Enhancing an artefact management system with traceability recovery features," in *ICSM*, 2004, pp. 306–315.

[45] P. Rempel, P. Mäder, and T. Kuschke, "Towards feature-aware retrieval of refinement traces," in *TEFSE@ICSE*, 2013, pp. 100–104.

[46] A. Dekhtyar, J. H. Hayes, S. K. Sundaram, E. A. Holbrook, and O. Dekhtyar, "Technique integration for requirements assessment," in *RE*. IEEE Computer Society, 2007, pp. 141–150.

[47] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *ICSE (1)*, 2010, pp. 95–104.

[48] D. Falessi, M. D. Penta, G. Canfora, and G. Cantone, "Estimating the number of remaining links in traceability recovery," vol. 22, no. 3, 2017, pp. 996–1027.

[49] C. Mills, J. Escobar-Avila, and S. Haiduc, "Automatic traceability maintenance via machine learning classification," in *ICSME*, 2018, pp. 369–380.

[50] 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 *ICSME*, 2019, pp. 103–113.

[51] 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 *ICSE*, 2020, pp. 873–885.

[52] C. M. Lüders, T. Pietz, and W. Maalej, "Automated detection of typed links in issue trackers," in *RE*, 2022, pp. 26–38.

[53] Y. Wang and H. Li, "Code completion by modeling flattened abstract syntax trees as graphs," in *AAAI*, 2021, pp. 14 015–14 023.

[54] H. Liu, Y. Wang, Z. Wei, Y. Xu, J. Wang, H. Li, and R. Ji, "Refbert: A two-stage pre-trained framework for automatic rename refactoring," in *ISSTA*, 2023, pp. 740–752.

[55] C. Lin, Z. Ouyang, J. Zhuang, J. Chen, H. Li, and R. Wu, "Improving code summarization with block-wise abstract syntax tree splitting," in *ICPC*, 2021, pp. 184–195.

[56] A. Mastropaolo, S. Scalabrino, N. Cooper, D. Nader-Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the usage of text-to-text transfer transformer to support code-related tasks," in *ICSE*, 2021, pp. 336–347.

[57] C. Niu, C. Li, V. Ng, J. Ge, L. Huang, and B. Luo, "Spt-code: Sequence-to-sequence pre-training for learning source code representations," in *ICSE*, 2022, pp. 2006–2018.