Back to the Basics: Rethinking Issue-Commit Linking with LLM-Assisted Retrieval

Anonymous Author(s)

Abstract

Issue-commit linking, which connects issues with commits that fix them, is crucial for software maintenance. Existing approaches have shown promise in automatically recovering these links. Evaluations of these techniques assess their ability to identify genuine links from plausible but false links. However, these evaluations overlook the fact that, in reality, when a repository has more commits, the presence of more plausible yet unrelated commits may interfere with the tool in differentiating the correct fix commits. To address this, we propose the Realistic Distribution Setting (RDS) and use it to construct a more realistic evaluation dataset that includes 20 open-source projects. By evaluating tools on this dataset, we observe that the performance of the state-of-the-art deep learning-based approach drops by more than half, while the traditional Information Retrieval method, VSM, outperforms it.

Inspired by these observations, we propose **EasyLink**, which utilizes a vector database as a modern Information Retrieval technique. To address the long-standing problem of the semantic gap between issues and commits, EasyLink leverages a large language model to rerank the commits retrieved from the database. Under our evaluation, EasyLink achieves an average Precision@1 of 75.91%, improving over the state-of-the-art by over four times. Additionally, this paper provides practical guidelines for advancing research in issue-commit link recovery.

Keywords

Issue-Commit Link Recovery, Software Traceability

1 Introduction

Software traceability involves establishing relationships between different software artifacts and is essential for safety-critical systems [10, 11]. A critical task in this domain is *issue-commit linking*, which connects issues, *i.e.,* bug reports, to the commits that resolve them [29], playing a vital role in software provenance. It also plays a key role for developers in assessing security risks [31, 42, 43] and gaining deeper insights about security flaws [38, 69].

Prior studies [3, 54] have revealed that many issue-commit links can be missing during the development of large-scale projects. Manually recovering these links is not only time-consuming but error-prone, even for experienced developers [55]. To address this, several studies [15, 32, 50, 55, 63, 71] have proposed learning-based approaches to automatically recover issue-commit links, achieving strong performance on datasets collected from open-source repositories. The evaluation method requires the tool to distinguish fix commits (*i.e.,* "true links") from plausible but non-fix commits (*i.e.,* "false links"). However, prior studies' evaluations often lack realism. ❶ Some studies use an unrealistic time window to select commits. For example, studies [4, 41, 55, 64] rely on a narrow 7-day time window to select potential fix commits, which may miss many true links. Another example involves studies [15, 50] that require the issue close time to select commits, but in practice, when commits are missing, the issue may not have a close time. ❷ Another limitation is the unrealistic false link distributions in prior evaluation datasets. Some [32, 55] use evaluation datasets where the number of false and true links is equal, an unrealistic assumption since, in reality, false links far outnumber true links [15]. Recently, Zhang *et al.* [71] addressed this issue by using an imbalanced dataset with a fixed number of false links per issue. Still, this method overlooks a crucial factor: a higher commit frequency results in a larger pool of plausible commits for the tool to differentiate from the actual fix commit, making the task inherently more difficult.

Therefore, we propose to construct a more realistic evaluation dataset under the **Realistic Distribution Setting** (RDS). We began with an in-depth analysis of the issues and commits from open-source repositories used in prior studies [15, 32, 37, 71]. According to our observation, after fetching all commits in each repository, approximately 97% of fix commits were made within one year from issue creation. These findings suggest that, in practice, the corresponding fix commit for an issue is likely to be found among the commits made within one year after the issue's creation. We thus include the genuinely linked commit as the true link and consider all non-fix commits made within one year of an issue's creation as candidate commits linked to the given issue as false links when constructing evaluation datasets.

After collecting issues and commits from 20 open-source projects previously analyzed in prior studies [15, 32, 37, 71], we first successfully replicate the performance of the state-of-the-art method, EALink [71], using the original evaluation method they reported. However, when we switched the experimental setup to the Realistic Distribution Setting, we find that the average Precision@1 of EALink dropped to 14.43%, surprisingly underperforming the Vector Space Model (VSM) [1], a traditional Information Retrieval (IR) technique, which achieves a Precision@1 of 46.59%, suggesting that there is great room for improving issue-commit linking using IR techniques. Meanwhile, as recent years have witnessed huge advances in the IR field, modern IR techniques that use embedding models to capture semantic-level similarities have demonstrated better performance than those traditional techniques evaluated in previous studies [15, 32, 71] that measure token-level textual similarities, such as VSM [1] and Latent Semantic Indexing (LSI) [36]. This also presents an opportunity to boost issue-commit linking performance and motivates our paper.

In this paper, we propose **EasyLink**, a novel method that integrates modern IR techniques and Large Language Models (LLMs) to enhance issue-commit linking performance in realistic settings by capturing deeper semantic relationships between issues and commits. EasyLink operates in two stages. The first stage fetches a set of commits that share similarities to a given issue, and the second stage reranks them by their relevance to the issue. Concretely, in the first stage, EasyLink leverages recent advances in Information Retrieval by using a modern vector database (optimized for high-dimensional similarity search) [16] to fetch the most similar commits for each issue. In the second stage, EasyLink prompts an LLM (GPT-4o [45]) to rerank the results. We use LLMs for their strong ability to capture the semantic relationship between issues and commits, bridging the semantic gap, as the most similar commit does not guarantee that it is the fix for the issue [15, 32, 71]. EasyLink achieves an average Precision@1 of 75.91%, outperforming EALink [71] by 4× in our realistic datasets and by 30% in Precision@1 on EALink's original evaluation setup. We also conduct an ablation analysis on each of EasyLink's two stages. Our results demonstrate that advances in IR, such as context-aware dense embeddings [40, 60, 67] and efficient vector search [16, 65], have been largely overlooked in the software traceability literature. Notably, even the out-of-the-box use of a vector database achieves a high average Precision@1 of 64.03%. Additionally, the strong performance of the reranking step, which improves Precision@1 by 11%, highlights the capability of LLMs to bridge the semantic gap between issues and commits [23, 50]. Finally, we discuss the lessons learned from our work, such as the need to consider modern IR baselines, for future research on software traceability. Our implementation, along with the evaluation dataset, has been anonymized and made available at [46].

This paper makes the following contributions:

(1) **Constructing a more realistic evaluation dataset:** To achieve a more realistic evaluation, we propose the Realistic Distribution Setting (RDS), which adjusts the number of candidate commits for generating false links according to the quantity of commits in the repository. This results in a dataset that more accurately reflects real-world practices.

(2) **A comprehensive evaluation benchmark:** We include the datasets from recent studies [15, 32, 37, 71]. Our benchmark includes 9,319 issues from 20 projects, with an average of 1,530 false links constructed per issue. To the best of our knowledge, this dataset is the largest in the literature.

(3) **Reevaluation of the state-of-the-art approach:** After successfully replicating the strong performance of EALink [71], the state-of-the-art approach, we switched the evaluation procedure under RDS, which offers a greater challenge due to a higher number of false links. On the same set of projects used in the evaluation of EALink, its average Precision@1 of 53.67% decreases to 28.05%, underperforming traditional IR baselines.

(4) **A new state-of-the-art for issue-commit linking:** We propose EasyLink, which leverages modern IR techniques, including an off-the-shelf database FAISS [16], and addresses the problem of semantic gap [23, 50] by prompting an LLM to rerank the retrieved results. In our evaluation, EasyLink improves over EALink in average Precision@1 from 14% to 76%.

The rest of the paper is organized as follows. Section 2 outlines the background of this work, including the limitations of existing work. Section 3 details the evaluation dataset construction under the Realistic Distribution Setting (RDS). Section 4 introduces EasyLink. Section 5 describes the experimental setup. Section 6 presents the experimental results. Section 7 discusses the lessons learned and threats to validity. Section 8 reviews related work. Section 9 concludes the paper. Finally, Section 10 presents the details of data availability.

An example of an issue, the incorrectly top-ranked commit, and the correct commit.

The input issue, with summary and description, is shown below.
Issue ID: NETBEANS-803

```
Summary:
nb-javac 11 upgrade in NetBeans
Description:
Should cover below tasks in NetBeans - nb-javac
     11 testing :
Run tests for modules java.completion, java.
     editor, java.editor.base, java.hints, java.
     source, java.source.base, lib.nbjavac
Update libs.javacimpl and libs.javacapi jars,
     upload updated nb-javac jars
Upload nb-javac module jars in update center
```

The top-ranked commit message in the initial results is shown below but is **incorrect**.
Commit ID: 4fd115aeae3b8423de9ced22d52914e60a1c5800.

```
Updation for external nb-javac jar in libs.
     javacapi and
libs.javaimpl modules with nb-javac jar for jdk
     -12
```

The **correct** commit message is shown below.
Commit ID: 3055661e4dd1c7d587012c917cbec31b27ae9e34

```
Uptake nb-javac 11 jars for java tests runtime
```

Figure 1: Example of an issue with the incorrect top-ranked commit and the correct commit. An issue-commit linking approach has to bridge the semantic gap and distinguish the correct commit from similar ones in a potentially large set.

2 Background

2.1 Issue-Commit Linking Recovery

Issue-commit links play an essential role in maintaining software traceability, supporting critical tasks such as impact analysis [2, 25, 48], regression testing [39], and project management [47]. Due to the high cost of manually maintaining these links, they are often incomplete [27], highlighting the need for automated methods. Automatic issue-commit linking considers a large set of commits that are made in the period after each issue's creation and identifies the right commits that address the issue. Figure 1 shows an example that demonstrates the challenge of automatically linking issues to commits. It presents an issue along with two commits. The first commit shares more matched words and has the highest similarity to the issue, but it is not the fix, while the second commit is the correct fix commit. An issue-commit linker that considers keyword counts would incorrectly prioritize the first commit as there are a greater number of matches of the keyword `nb-javac`. While both commits mention `nb-javac`, the top-ranked commit updates the JAR for JDK-12, whereas the issue specifies that nb-javac 11 should be upgraded, tested, and deployed. This highlights two challenges. First, it shows the need for approaches that go beyond surface-level similarities. Second, it demonstrates the sensitivity of the evaluations of their effectiveness to the number of commits that share keywords or resemble the ground-truth commit.

Many automatic issue-commit link recovery methods have been proposed. Some studies employ traditional feature- and rule-based methods [4, 41, 58, 68], which rely on predefined heuristics such as keyword matching or the recency of the commits. However, these heuristics tend to be inadequate [5]. Some approaches [30, 37, 50, 63, 64] adopt traditional machine learning techniques, such as support vector machines, to reduce the reliance on manual rules. Recently, deep learning methods [23, 55, 70] have demonstrated improved performance. A key challenge of the task is the semantic gap that exists between issues and commits [15, 32, 71]. Prior work [32] attempted to address this issue using BERT-based methods, such as CodeBERT [18], which leverage contextual understanding. The state-of-the-art method, EALink [71], employs knowledge distillation to transfer knowledge to a smaller model and utilizes multi-task learning to improve both accuracy and efficiency.

2.2 Limitations in Evaluation

The experiments in prior studies evaluate approaches based on their ability to distinguish the true link from false links constructed using other commits. However, these studies use unrealistic methods to construct false links and the evaluation dataset.

Unrealistic Time Window Selection: Some works [4, 41, 55, 64] assume that fix commits fall within a 7-day time window before or after the issue's create/update/close time or the comment create time, treating all other commits in this period as non-fix commits. However, this approach is inadequate when fixes take a longer time, requiring tools to distinguish the correct fix commit from more plausible commits. From our preliminary analysis of selected projects, we found that only 59% of issues had a corresponding fix commit within seven days of their creation. Additionally, some works [15, 50] require the issue's close time to select commits. Under a practical scenario, the issue fix/close date is unknown—precisely also why issue-commit recovery is necessary—and approaches need to distinguish the fix commit while the issue remains open. Therefore, selecting commits based on the issue's fix/close date may also lack realism.

Unrealistic False Link Distribution: When constructing the evaluation dataset, some works [32, 37, 55] use a balanced dataset (*i.e.,* an equal number of false links and true links) to evaluate their tools. This evaluation setting is unrealistic because, in reality, false links outnumber true links [15]. Zhang *et al.* [71] address the balanced dataset limitation by constructing a fixed number of 99 false links per issue. While this results in more false links than true links, it still overlooks that the number of potentially linked commits depends on the number of commits made in the same time period as the ground-truth link. Moreover, as shown in Figure 2, when sampling unrelated commits to construct the false links, prior work [71] only selects commits from other ground-truth links. In other words, they construct false links for an issue by connecting it only to other commits that are already linked to other issues. In practice, an approach has to consider all commits from the same

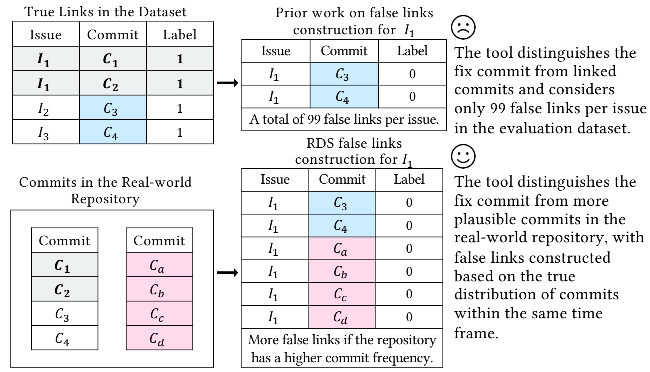

Figure 2: Illustration of the evaluation limitation in prior work. C_1 and C_2 are the fix commits of issue I_1. False links will be constructed for I_1. C_3 and C_4 are commits already linked to the issue in the true links dataset. C_a, C_b, C_c, and C_d are commits present in the repository but not in the true links dataset, and they are ensured not to be the fix commit of I_1.

time period, regardless of whether they are linked to specific issues. This is a superset of commits compared to the ground-truth links. In Section 3, we show that the number of commits made within a one-year time frame is at least 80% larger than the constant number of 99 commits considered by Zhang *et al.* [72]. As a result, the evaluation setups of prior work may not adequately reflect how these approaches would be used in practice.

3 Evaluation Dataset Construction Under the Realistic Distribution Setting

This section describes our new evaluation setting, **Realistic Distribution Setting** (RDS), for constructing a more realistic evaluation dataset. By adaptively constructing false links for an issue based on the commits in the period after each issue creation, aligning with the repository's development activity, our method addresses the limitations of prior evaluations.

We combine datasets from four recent studies [15, 32, 37, 71] into a unified benchmark covering 20 projects, ensuring broader coverage and a consistent benchmark for comparison. Evaluations in the previous studies [32, 37, 71] did not use a shared benchmark, and each of their datasets had only up to 12 projects [37]. Our dataset fills the need for a large, shared benchmark.

3.1 Ground Truth Dataset Selection

We selected datasets from four recent studies, covering 20 open-source software projects over a span of 20 years. We include the same issues and true links from these datasets, resulting in a total of 9,319 issues in the benchmark. For datasets with incomplete information detected through a manual check, we re-fetched the necessary data, such as issue comments, code diffs, and committed files (*i.e.,* the source code files after applying the commits).

- **From Zhang *et al.*'s dataset [71]**, we include the issues from all six projects, *Ambari*, *Calcite*, *Groovy*, *Ignite*, *Isis*, and *Netbeans*, in our benchmark.

- **From Dong *et al.*'s dataset [15]**, we include the issues from five projects, *Pig*, *Maven*, *Infinispan*, *Drools*, and *Derby*. This dataset also includes *Groovy*, which is already included in our benchmark from Zhang *et al.*'s dataset [71]. As they were missing issue comments, code diffs, and committed files, we refetched these data.
- **From Lin *et al.*'s dataset [32]**, we include the issues from all three projects: *Pgcli*, *Flask*, and *Keras*. As the issue comments and committed files were missing, we refetched them.
- **From Mazrae *et al.*'s dataset [37]**, we include the issues from six projects, *Beam*, *Flink*, *Freemarker*, *Airflow*, *Arrow*, and *Cassandra*, out of 12 projects. The six other projects from this dataset are already included in Zhang *et al.*'s dataset [71]. As this dataset provides only postprocessed data, we re-fetched the original issue summaries, issue descriptions, issue comments, commit messages, code diffs, and committed files.

3.2 Preparing Commits

For constructing false links, the set of all commits in each repository is required. First, we clone the GitHub repository locally, which enables faster access to commit data without API requests. From the cloned repository, all commit IDs are obtained using the `git log` command. Each commit is then processed to retrieve its metadata, including the parent commit IDs, author, committer, commit time, and commit message, using `git show`. Additionally, the list of modified file paths for each commit is identified, and their content at the specific commit state is retrieved. The corresponding code diffs are also extracted and stored. Table 1 presents the number of commits fetched for each project in the column "#Commits".

The following information is fetched during the process:

- **Commit ID**: A unique hash value assigned to each commit, serving as its identifier.
- **Parent Commit IDs**: The hash(es) of the immediate predecessor commit(s) of the current commit.
- **Author**: The person who originally wrote the changes.
- **Committer**: The person who applied the changes to the repository.
- **Commit Time**: The commit's creation timestamp.
- **Commit Message**: The description of the commit.
- **Changed Files**: List of paths of the files modified in the commit.
- **Code Diffs**: The changes introduced by the commit.
- **Committed Files**: Source code files after applying the commit.

Note that while our work only requires the commit ID, commit time, and commit message, we collect all commit information since other approaches may utilize this additional data.

3.3 Constructing False Links

For a more realistic evaluation dataset, the number of false links should match the actual number of commits that may be viable for linking to each issue in practice. For each issue in the evaluation dataset, under the Realistic Distribution Setting (RDS), false links will be adaptively constructed to better align with the repository's level of activity. We include all commits submitted in the time window within which the true commit can appear. To determine the size of the time window, we perform an analysis and find that approximately 97% of all issues have their ground truth commit submitted within one year of their creation (7-day time window: 59%, 30-day time window: 77%, 6-month time window: 92%). This observation aligns with previous studies analyzing bug reports (*e.g.*, Rodrigues *et al.* [52], Zhang *et al.* [72]). Consequently, we apply a one-year time window to select candidate commits from the commits pool (as described in Subsection 3.2) for each issue. Specifically, for a true link denoted as $t_i = \{I_i, C_i\}$, where I_i represents the issue and C_i the commit in the true link, the candidate commit C_j is determined using Equation 1. If the commit time of C_j is within one year after the creation time of I_i, we treat C_j as a candidate commit. By linking C_j to I_i, we generate the false link $f_i = \{I_i, C_j\}$.

$$\begin{aligned} \text{is\_candidate}(I_i, C_j) = \mathit{created}(I_i) &\leq \mathit{committed}(C_j) \\ \wedge\ \mathit{committed}(C_j) &\leq \mathit{created}(I_i) + \epsilon, \\ \epsilon &= \text{one year}. \end{aligned} \quad (1)$$

Algorithm 1: False Links Construction under RDS

```
   Input: test_set_true_links      ▷ Contains true issue-commit
          pairs labeled as 1
   Input: commits_pool             ▷ Contains all fetched commits
          (Subsection 3.2)
   Result: evaluation_dataset
1  Function EvalSetGen(test_set_true_links, commits_pool):
2      evaluation_dataset ← ∅
3      sampled_issue_id_list ←
         test_set_true_links[issue_id].unique()[:1000]
4      for each issue_id in sampled_issue_id_list do
5          true_links ← test_set_true_links[issue_id]
6          issue ← true_links[0].issue_info
7          candidate_commits ← { commit ∈ commits_pool |
8              commit.time ≥ issue.create_time and
9              commit.time < issue.create_time + 1 year and
10             commit ≠ issue.fix_commit }
11         false_links ← {(issue, commit, label=0) |
12             commit ∈ candidate_commits }
13         evaluation_dataset.append(false_links)
14         evaluation_dataset.append(true_links)
15     return evaluation_dataset
```

Algorithm 1 details the construction of the evaluation dataset. We first split the entire ground-truth dataset into training and test sets following a 4:1 ratio [15, 55, 71]. The inputs to the algorithm are the true links from the test set and the commits pool for the project, which is prepared using the process described in Subsection 3.2. First, following the method used by EALink [71], we randomly sample up to 1,000 unique issue IDs from the test set, forming the list *sampled_issue_id_list*. If fewer than 1,000 unique issues are available, all issues are included. Then, for each *issue_id* in this list, we extract the true links for that issue from the test set. We use the term "links" (plural) because an issue may be linked to more than one commit [71]. In line 6 of the algorithm, we extract only the issue information for the issue being processed (including issue ID, summary, description, etc.), denoted as *issue*. Next, as shown in lines

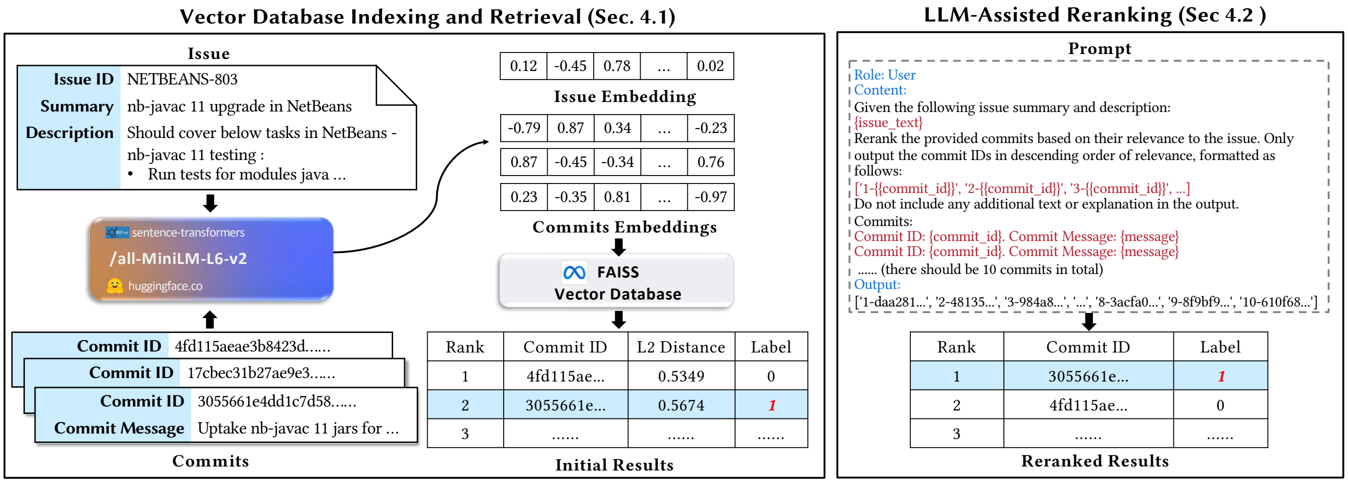

Figure 3: Overview of EasyLink. EasyLink consists of two key steps—the first step utilizes a vector database to retrieve initial ranked results, and the second step prompts an LLM to rerank the results.

Table 1: Statistics of the benchmark

Project name	#Commits	#Unique issue_id	Average # of false links per issue
Ambari⋆	24809	1000	3978.26
Calcite⋆	5889	551	516.31
Groovy⋆	20862	1000	1064.79
Ignite⋆	28869	1000	2611.81
Isis⋆	24945	652	1265.91
Netbeans⋆	10502	159	1279.68
Derby†	8040	43	363.93
Drools†	16585	182	630.74
Infinispan†	17078	399	1110.68
Maven†	14685	61	377.15
Pig†	3675	45	197.44
Flask‡	5353	151	357.88
Keras‡	11248	111	454.23
Pgcli‡	2364	105	356.92
Airflow§	27123	961	1882.61
Arrow§	16947	1000	2058.99
Beam§	43595	865	6096.72
Cassandra§	29867	25	2560.84
Flink§	35767	1000	3251.68
Freemarker§	2492	9	179.00

Note: Each project is annotated with a superscript representing its original dataset source: ⋆ Zhang *et al.* [71], † Dong *et al.* [15], ‡ Lin *et al.* [32], § Mazrae *et al.* [37]

7 to 10 of the algorithm, we filter the *candidate_commits* such that each commit's creation time is no earlier than the issue's creation time and strictly earlier than one year after the issue's creation time. Additionally, the commit should not be the fix commit of the issue. We then construct the *false_links* by pairing the *issue* with each commit in *candidate_commits* and labeling the pair as 0. Finally, we append both the *true_links* and the *false_links* for the issue to the evaluation dataset and then proceed to process the next issue. Note that this algorithm is executed separately for each project, as each project has its own test dataset and commits pool.

Table 1 shows the statistics of the evaluation dataset. The dataset includes the same issues as used in the evaluation of prior studies [15, 32, 37, 71]. Each issue has an average of 1,530 false links, with a minimum of 179 (Freemarker). In contrast, for each issue, EALink [71] constructs a constant number of 99 false links per issue, which is substantially smaller than the number of commits within a one-year time frame from issue creation. Note that an issue may be fixed by one or more commits [71]. In our evaluation dataset, 17% of issues were fixed by multiple commits.

4 The EasyLink Approach

This section details EasyLink. Figure 3 shows overview of EasyLink, which consists of two stages: the first stage uses a vector database for scalable retrieval to fetch commits ranked by their similarity to the issue and the second stage prompts an LLM to rerank them by their relevance to the issue. We elaborate the key steps, vector database indexing and retrieval, and LLM-assisted reranking below.

4.1 Vector Database Indexing and Retrieval

The process begins by embedding the commit messages using an embedding tool, retaining the commit ID as metadata. Similarly, the issue summary and description are concatenated and embedded together, with the issue ID retained as metadata. To generate the embeddings, we utilize the Sentence-Transformers library [51] with the `all-MiniLM-L6-v2` model, a lightweight and efficient transformer based on Microsoft's MiniLM architecture [67]. We chose this model because it is the most downloaded and most liked sentence-similarity model on Hugging Face, indicating strong community trust and widespread adoption. This model employs self-attention distillation to capture contextual information effectively while maintaining computational efficiency. Given an input text T, the embedding process can be defined as follows:

$$\mathbf{E} = \text{MiniLM}(T) \tag{2}$$

where $\mathbf{E} \in \mathbb{R}^{384}$ represents the output embedding vector in a 384-dimensional space.

After generating embeddings, we use a vector database for indexing and retrieval. We use FAISS [16]—an off-the-shelf library designed for efficient nearest neighbor (NN) retrieval in high-dimensional spaces. We believe that switching FAISS with another vector database with comparable capability would produce similar results.

To retrieve commits from the vector database, we compute the squared L2 (Euclidean) distance between the query embedding and the stored embeddings, as we use the vector database through LangChain [9], where L2 distance is used by default. The computation is given by the following equation:

$$distance(\mathbf{q}, \mathbf{e}_i) = |\mathbf{q} - \mathbf{e}_i|_2^2 \tag{3}$$

where e_i is the element of the commit message embeddings $\mathbf{E} = \langle \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n \rangle$ that are indexed by the vector database for efficient access, and $\mathbf{q}$ is an issue embedding. A shorter distance between the query issue and the commit message indicates greater similarity, enabling the ranking of commit messages and the generation of a list of candidate commits.

4.2 LLM-Assisted Reranking

Due to the semantic gap between issues and commits [15, 32, 71], retrieving the most similar commit does not guarantee finding the one that fixed the issue. While the correct commit may have been retrieved, it could be obscured by incorrect commits that exhibit greater similarity to the issue. To address this, we rerank the retrieved commits using a large language model (LLM).

In this phase, the top-k commits from the initial retrieval are provided to the LLM with a structured prompt, as shown in Figure 3. The prompt includes the issue text (summary and description), along with each commit's ID and message. We also specify an output format instructing the LLM to return a reranked list of commit IDs. The LLM will analyze these commits and produce a reranked list based on contextual understanding and issue-commit relevance. Outputs that do not match the expected format (0.1% of the time) default to the initial retrieval results.

In detail, we adopt a zero-shot approach guided by prompts with ChatGPT (*gpt-4o*), following recent studies [20, 26, 66], which demonstrate *gpt-4o* advanced capabilities in understanding complex textual relationships. The parameter k controls the additional cost incurred for increasing precision. While it is possible to rerank all fetched commits for higher precision, this requires a longer prompt for the LLM, requiring more computation resources. For the value of k, we select k=10 for its balance of precision and efficiency. We later show that this allows a high Precision@1 without incurring a high cost. This will be discussed in Subsection 6.3.

5 Experimental Setup

5.1 Research Questions

This work aims to answer the following research questions (RQs):

RQ1: How does the state-of-the-art tool perform on a realistic evaluation dataset? This question investigates the performance of the state-of-the-art EALink [71] on the evaluation dataset constructed under the Realistic Distribution Setting (RDS). First, we replicate the successful performance of EALink on their original evaluation dataset. Next, we investigate the performance of EALink after expanding the evaluation onto a larger benchmark. Afterwards, to understand the sensitivity of its performance to the evaluation dataset, we investigate how much the performance of EALink changes when evaluated under Realistic Distribution Setting and compare it with the traditional IR method, VSM.

RQ2: How does EasyLink perform on the same realistic evaluation dataset? This research question is concerned with the effectiveness of EasyLink. We assess EasyLink in terms of both its ability to distinguish true links from false links and its efficiency. We analyze the sensitivity of the performance of EasyLink to the evaluation setup by comparing its performance on the different evaluation methods.

RQ3: Does EasyLink's performance change under different configurations? This question aims to investigate the effect of different settings of EasyLink. We explore different embedding models for the vector database and various k value settings for the LLM-assisted stage to assess whether these changes will significantly affect EasyLink's performance.

5.2 Experiment Setting

5.2.1 Hardware Configuration. The experiments were conducted on a machine equipped with two Intel(R) Xeon(R) Platinum 8480C CPUs @ 3.80GHz, 2.0 TiB of main memory, and one NVIDIA H100 80GB HBM3 GPU.

5.2.2 LLM Setup. We utilized the ChatGPT model GPT-4o provided by OpenAI, specifically the `gpt-4o-2024-11-20` version, with its default configuration. The temperature was set to the default value of 1.0, and the model was sampled once per query. The input token size was limited to a 128k token context window per API constraints, with no additional adjustments or fine-tuning.

5.2.3 Baseline Description. We use EALink[1] [71] as the baseline, which is a state-of-the-art tool that outperforms T-BERT [32] and DeepLink [55]. It distills knowledge from CodeBERT [18] into a smaller model, fine-tuned with multi-task contrastive learning. We follow its original methodology, using the provided code and the same hyperparameters. As a fundamental baseline, we also run VSM[2] [57], which is widely used in other studies [15, 32, 71].

5.3 Evaluation Metrics

We adopt the same metrics used by Zhang *et al.* [71], which use standard metrics for information retrieval tasks [49, 56]: Precision@k (P@k), Normalized Discounted Cumulative Gain (NDCG@k), Mean Reciprocal Rank (MRR), and Hit Ratio (Hit@k) for evaluation. We also include Recall@k, which was overlooked in prior studies.

- **Precision@k** evaluates the proportion of relevant commits (*i.e.,* commits that belong to the correct issue-commit links) within the top k retrieved results:

$$\text{Precision@k} = \frac{1}{|Q|} \sum_{i \in Q} \frac{\text{Rel}_i}{k}, \tag{4}$$

[1]We use EALink provided code and data from https://github.com/KDEGroup/EALink.
[2]We implemented VSM using the Gensim library (https://pypi.org/project/gensim)

where Q is the query set, $|Q|$ its size, and Rel_i the number of correctly linked commits in the top k results for query i.

- **Hit@k** measures the likelihood that at least one correct commit appears within the top k retrieved results:

$$\text{Hit@k} = \frac{1}{|Q|} \sum_{i} \mathbb{I}(\text{Rank}_i \leq k), \tag{5}$$

where $\mathbb{I}(\cdot)$ returns 1 if the highest-ranked relevant commit for query i is within the top k, and 0 otherwise. Hit@1 is equivalent to Precision@1.

- **Recall@k** evaluates the proportion of relevant commits retrieved within the top k results:

$$\text{Recall@k} = \frac{1}{|Q|} \sum_{i \in Q} \frac{\text{Rel}_i}{\text{TotalRel}_i}, \tag{6}$$

where Q is the query set, $|Q|$ its size, Rel_i the number of retrieved relevant commits, and TotalRel_i the total relevant commits for query i.

- **MRR** (Mean Reciprocal Rank) evaluates how early the first relevant commit appears in the ranked list for each query:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{Rank}_i}, \tag{7}$$

where $|Q|$ is the number of queries, and Rank_i is the position of the first correctly linked commit for query i. A higher MRR indicates earlier retrieval of relevant commits.

- **NDCG@k** (Normalized Discounted Cumulative Gain) assesses how well relevant commits are ranked within the top k retrieved results:

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

where Z_k is a normalization factor ensuring the ideal ranking achieves a value of 1. The term r_i represents the relevance score of the commit at position i ($r_i = 1$ for a correct match, 0 otherwise).

6 Results

6.1 RQ1: How does the state-of-the-art tool perform on a realistic evaluation dataset?

To ensure we replicate EALink [71] correctly and perform a fair comparison, we ran EALink on both the evaluation dataset constructed using its original method and the dataset created under the Realistic Distribution Setting (RDS), then compared the results. In the original experiments of Zhang *et al.*, EALink was trained using a balanced dataset and then tested on an imbalanced dataset. We reused the same code for training.

Following EALink's original evaluation [71], we first used their false link generation method to construct an evaluation dataset using projects provided by them: Ambari, Calcite, Groovy, Ignite, Isis, and NetBeans. For any given issue, it randomly samples 99 of the commits in the ground truth test dataset to construct false links. Next, with this evaluation dataset, which we refer to as the original evaluation dataset, we evaluated EALink. EALink obtains an average Precision@1 of 53.67%, which matches the 53.90% Precision@1 reported in the paper, indicating that our replication was successful. Then, we expanded the evaluation to 20 projects (introduced in Subsection 3.1). The results, shown in the first column of Table 2, indicate an average Precision@1 of 32.52%.

Table 2: Performance of EALink on evaluation datasets constructed using different methods

	False Links=99	RDS
P@1 (Hit@1)	32.52	14.43 (↓ 55.63%)
P@10	7.08	3.64 (↓ 48.59%)
Hit@10	59.46	30.76 (↓ 48.27%)
Recall@10	56.68	28.06 (↓ 50.49%)
MRR	41.16	20.21 (↓ 50.90%)
NDCG@1	20.52	9.10 (↓ 55.65%)
NDCG@10	30.84	15.52 (↓ 49.67%)

Note: Results are averaged over 20 projects. The column False Links=99 shows EALink's performance on a dataset constructed with a fixed 99 false links per issue, while the column RDS shows its performance on a dataset constructed under Realistic Distribution Setting (RDS). The parentheses in the RDS column show the percentage change in performance.

Then, we ran EALink on the evaluation dataset constructed under the Realistic Distribution Setting. Unlike the original evaluation dataset of Zhang *et al.* [71], which had a fixed number of 99 false links per issue, our evaluation generates more false links for repositories with a larger number of commits. This results in an average number of false links per issue of 1529.78. In the second column of Table 2, we present the results of running the tool on our evaluation dataset, which shows a significant performance drop compared to the original evaluation dataset. Across the 20 projects, EALink's Precision@1 decreases to 14.43%, a decline of 55.63%. Similarly, Hit@10 drops from nearly 60% to about 30%, reducing by half. On the six projects provided by EALink (including Ambari, Calcite, and four others), the average Precision@1 decreases from 53.67% to 28.05%. Additionally, we evaluated the traditional IR method, VSM, on the same evaluation dataset. Its results are presented in Table 4. Surprisingly, EALink underperforms the traditional IR method, VSM. VSM achieves a Precision@1 of 46.59% and demonstrates better performance across all other metrics. These results suggest that the use of VSM would be preferred over EALink in a realistic setting.

Answer to RQ1: We constructed an evaluation dataset under the Realistic Distribution Setting (RDS), which considers more false links per issue when there is higher commit activity during a given period, better reflecting the practical use of an issue-commit link recovery technique on a repository. Under this evaluation, the average Precision@1 of the state-of-the-art tool declines to 14.43% compared to a VSM baseline with a Precision@1 of 46.59%.

6.2 RQ2: How does EasyLink perform on the same realistic evaluation dataset?

6.2.1 Comparison of Effectiveness. To ensure a fair comparison, we first evaluated the vector database and EasyLink on the dataset constructed using the EALink method [71], which generates 99 false links per issue and includes the six projects provided by EALink.

Table 3: Comparison of linking effectiveness on the original dataset with a constant 99 false links constructed per issue. This comprises the six projects provided by Zhang *et al.* [71] (Ambari, Calcite, Groovy, Ignite, Isis, and NetBeans)

	EALink	Vector DB	EasyLink
P@1 (Hit@1)	53.67	81.59	**89.58**(↑ 66.90%)
P@10	8.94	**11.08**	**11.08**(↑ 23.93%)
Hit@10	72.56	**92.44**	**92.44**(↑ 27.39%)
Recall@10	69.03	**89.43**	**89.43**(↑ 29.55%)
MRR	60.13	85.57	**90.80**(↑ 51.00%)
NDCG@1	33.86	51.48	**56.52**(↑ 66.92%)
NDCG@10	41.33	56.07	**58.03**(↑ 40.41%)

Note: The parentheses in the EasyLink column show the improvements over EALink. Bold numbers indicate the highest performance. All values are in %.

Table 4: Comparison of linking effectiveness on the dataset constructed under RDS

Metric	EALink	VSM	Vector DB	EasyLink
P@1 (Hit@1)	14.43	46.59	64.03	**75.91** (↑ 426.1%)
P@10	3.64	8.70	**10.82**	**10.82** (↑ 197.3%)
Hit@10	30.76	70.94	**85.35**	**85.35** (↑ 177.5%)
Recall@10	28.06	67.22	**81.48**	**81.48** (↑ 190.4%)
MRR	20.21	55.10	71.59	**79.92** (↑ 295.4%)
NDCG@1	9.10	29.39	40.40	**47.89** (↑ 426.3%)
NDCG@10	15.52	39.45	49.37	**52.34** (↑ 237.2%)

Note: Results are averaged over 20 projects (%). The parentheses show the improvements over EALink. Since P@10, Hit@10, and Recall@10 only assess whether the true commits are included within the top 10 results, without considering its exact rank, they are the same for both the vector database and EasyLink. Bold numbers indicate the highest performance.

The results, shown in Table 3, indicate that EasyLink achieves a Precision@1 of 89.58%, significantly outperforming EALink's 53.67%. The vector database method also outperforms EALink, achieving a Precision@1 of 81.59%.

Furthermore, we compared them using our more realistic evaluation dataset. As presented in Table 4, the vector database approach achieves an average Precision@1 of 64.03%, a significant improvement over EALink's 14.43%. With LLM-assisted reranking, EasyLink further enhances Precision@1 by an average of 11.88%, reaching 75.91%, representing a 426.1% increase compared to EALink. Following the Mann-Whitney U test [35], the improvement of EasyLink over EALink in every metric is statistically significant (p-value < 0.01) and exhibits a large effect size [12] (Cohen's D $>$ 0.8). These results demonstrate that EasyLink is effective in issue-commit link recovery and that LLMs can be effectively leveraged to enhance performance.

6.2.2 *Comparison of Efficiency.*

Efficiency is an important aspect of software traceability, especially when used in large-scale industrial settings [10, 61]. Therefore, we calculated the training and testing times required for three approaches—EALink, the vector database, and the EasyLink—to compare their efficiency. The results, shown in Table 5, indicate that for the 20 projects, EALink requires 100.68 hours for training, with an average of 4.39 seconds per link in the training set. However, approaches using a vector database do not require training on a specific issue-commit link dataset, making them easier to use and more practical with less human effort.

Table 5: Comparison of training and testing time cost

	EALink	Vector DB	EasyLink
Train (Total / Per Link)	100.68h / 4.39s	N/A	N/A
Test (Total / Per Issue)	17.78h / 6.87s	2.60h / 1.01s	17.76h / 6.86s

Note: "Total" indicates the overall time for all 20 projects. "Per Link" (training) is the average time per link, computed as total training time divided by the number of training links. "Per Issue" (testing) is the average time per issue, based on total testing time divided by the number of evaluation issues. Testing times for Vector DB and EasyLink include embedding, indexing, and retrieval.

Regarding testing time, EALink requires a total of 17.78 hours for the 20 projects, averaging 6.87 seconds per issue, while the vector database approach takes only 2.60 hours in total, averaging 1.01 seconds per issue. These differences are statistically significant according to the Mann-Whitney U test [35] (p-value < 0.01) and demonstrate a large effect size (Cohen's D > 0.8). Additionally, the LLM-assisted method (*i.e.,* EasyLink) takes 17.76 hours for testing, which is slightly faster than EALink but achieves a significantly higher performance. These results demonstrate that the use of vector database is an efficient solution. It eliminates the need for pretraining models and reduces testing time. By incorporating an LLM reranking step, EasyLink achieves a high precision while eliminating the need for training models.

Answer to RQ2: Our results show that on the realistic evaluation dataset, EasyLink achieves a Precision@1 of 75.91%, outperforming EALink (which achieves 14.43%) by 426.1%. Moreover, using the vector database-based method eliminates the need for pretraining on task-specific data, making it a more efficient solution.

6.3 RQ3: Does EasyLink's performance change under different configurations?

6.3.1 *Vector Database: Evaluating Different Embedding Models.*

We evaluated two additional embedding models for use in the vector database. The first model, `all-mpnet-base-v2`, built upon Microsoft's MPNet architecture [60], is a sentence transformer and the second-most-downloaded sentence-similarity model on Hugging Face (the most popular is used in EasyLink). The second model, `text-embedding-ada-002`, is the default model for OpenAI embeddings [40], which is a larger model designed for higher precision.

Table 6 presents the results comparing three embedding models. The comparison between MiniLM (used in EasyLink) and MPNet shows that the model we selected in our tool requires less time while achieving relatively better performance. Although OpenAI embeddings provide better performance—with Precision@1 increasing from 64.03% to 72.16% (an improvement of 8.13%)—they require 29.61 hours to complete the experiment, approximately 14 times longer than the MiniLM model. Given the high computational cost, MiniLM is a more practical option.

Table 6: Comparison of linking effectiveness while varying the choice of embedding method

	MiniLM	MPNet	OpenAI
P@1 (Hit@1)	64.03	61.22	72.16
P@10	10.82	10.37	11.09
Hit@10	85.35	82.39	88.64
Recall@10	81.48	78.58	84.73
MRR	71.59	69.02	78.40
NDCG@1	40.40	38.65	45.53
NDCG@10	49.37	47.62	52.67
Test Time (hour)	2.60h	3.10h	29.61h
Model Size	22.7M params	109M params	Cloud-based

Note: Results are averaged over 20 projects. All values are in %, except for test time and model size.

6.3.2 LLM-Assisted Reranking: Exploring Different Top-k Values. To further analyze the effect of the k setting, we conduct experiments with different top-k values, specifically $k = 5, 10, 15, 20$. We conducted this experiment across 20 projects using the realistic evaluation dataset. The results in Figure 4 show that increasing k—meaning reranking a larger set of top results—generally leads to better performance, specifically higher Precision@1 and NDCG@1. However, this improvement comes with a significant increase in time cost and requires processing more input tokens, leading to a higher cost in invoking OpenAI API. While setting $k = 5$ results in relatively lower performance, increasing k to 10, 15, or 20 does not lead to significant performance gains. Considering performance, time, and resource costs, we find that $k = 10$ provides the best trade-off. Additionally, compared to the straight line in the figure representing the Precision@1 of EALink's when run on the same evaluation dataset, our approach outperforms EALink across different k values. This shows that its performance does not rely on tuning k.

> **Answer to RQ3**: For vector database retrieval, using different embedding models with similar runtime costs yields comparable results. In the LLM-assisted reranking stage, increasing the k value consistently improves performance over the baseline but increases test time cost.

7 Discussion

7.1 Lessons learned and Implications

Keep your feet on the ground – evaluation should match practice. Our results suggest that the performance of issue-commit linking approaches is sensitive to their evaluation setups, which emphasizes the importance of the evaluation of proposed techniques to reflect the conditions under real-world practices. Future research should use evaluations that better reflect a practical usage scenario.

Back to the basics – IR techniques are strong baselines. Our results showed that the VSM baseline, a dated IR model [1] proposed by Wu *et al.* [68] for issue-commit linking, was effective. This is consistent with the "Easy over hard" principle advocated by Fu and Menzies [19]. This finding has practical implications – practitioners

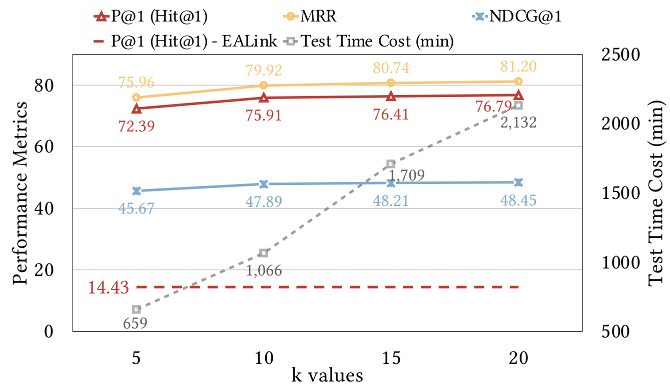

Figure 4: Effect of varying k in EasyLink during reranking: A higher k slightly raises performance (left axis: P@1, MRR, NDCG@1) while significantly increasing test time cost (right axis).

may find that the use of simple and fast approaches match the performance of more complex approaches.

Don't forget your roots – updating baselines. Studies on issue-commit linking have included traditional IR baselines, such as the Vector Space Model (VSM) [57], Latent Dirichlet Allocation (LDA) [6], and Latent Semantic Indexing (LSI) [13]. However, recent studies continue to rely on traditional methods as baselines without considering recent advances in the IR literature. In our experiments, the out-of-the-box use of a modern baseline, the vector database FAISS [16], already achieves a strong performance. This shows that vector databases provide an effective retrieval approach for issue-commit linking, highlighting the need to update baselines to reflect the latest advancements in IR methods. We call for the need for newly proposed methods to be evaluated with baselines that are continuously updated to accurately measure real progress.

From good to great – effectiveness of LLMs in retrieval refinement. Our experiments demonstrate that a reranking step using LLMs is highly effective in overcoming the semantic gap, consistent with findings in information retrieval systems [21, 62, 74]. To assess whether LLM-based reranking can also enhance the performance of EALink [71], a deep learning-based method, we conducted experiments on our realistic evaluation dataset, including 20 projects. As shown in Table 7, with LLM reranking assistance, we observe a Precision@1 improvement of 80.45%, increasing from 14.43% to 26.04%. This demonstrates the LLM's effectiveness in refining initially imprecise results. Future work can explore the optimization of LLM prompts and apply domain-specific fine-tuning to further improve the refinement step.

7.2 Threats to Validity

Threats to Internal Validity. Threats to internal validity refer to errors in our experiments or implementation issues. To avoid implementation errors, we replicated the baseline tool, EALink [71], using its publicly available code. We ensured that we replicated

Table 7: Comparison of EALink and EALink+LLM

Metric	EALink	EALink+LLM	Improvement
P@1 (Hit@1)	14.43	26.04	↑ **80.45**
P@10	3.64	3.64	-
Hit@10	30.76	30.76	-
Recall@10	28.06	28.06	-
MRR	20.21	28.55	↑ **41.26**
NDCG@1	9.10	16.96	↑ **86.37**
NDCG@10	15.52	18.79	↑ **21.07**

Note: Results (%) are averaged over 20 projects on our dataset constructed under the Realistic Distribution Setting (RDS).

its previously reported results before extending the experiments. Therefore, the threats to internal validity are minimal.

Threats to Construct Validity. A potential threat to construct validity is the selection of evaluation metrics. We use widely adopted metrics—Precision@k, Hit@k, MRR, and NDCG@k—from prior studies [15, 37, 55, 71] and information retrieval tasks [49, 56]. We also included Recall@k, a standard information retrieval evaluation metric, overlooked in prior issue-commit linking work. Consequently, we believe that any threat to construct validity is minimal.

Threats to External Validity. Threats to external validity refer to factors that might limit the generalizability of our findings. Our benchmark is the largest in the literature, which provides confidence that our findings are not specific to only a few projects. One threat is the number of times our experiments were performed. To mitigate the effect of randomness, we repeated our experiments five times. For all metrics, the average results exhibit standard deviations below 1%. Given the stability of the results, repeating the experiments would not yield different findings. As such, there are minimal threats to external validity.

8 Related Work

Traceability Link Recovery: Traceability link recovery methods create links between artifacts such as requirements, design documents, architecture models, and source code. Research has applied classic IR techniques [7, 17, 22, 34, 44]. Recent work [20, 53] utilized LLMs but also found that their level of effectiveness has still been unable to support practical automatic link recovery [20, 24].

Traditional Approaches for Issue-Commit Linking: Traditional approaches combine heuristics and expert annotation to link commits with bug reports. Bachmann *et al.* [4] used interactive heuristic linking. Wu *et al.* [68] filtered candidates by textual similarity, timing, and committer mapping. It also learned optimal thresholds from training data. Nguyen *et al.* [41] improved performance by adding code change analysis. Schermann *et al.* [58] leveraged developer identity, time proximity, and resource overlap. These methods often miss links [5].

Machine Learning-Based Approaches: Machine learning methods improve linking accuracy by reducing the need for handcrafted heuristics. Le *et al.* [30] enriched commit messages via code summarization. Sun *et al.* [64] refined feature extraction with non-source documents and code file filtering. Sun *et al.* [63] used positive-unlabeled learning to address limited labeled data. Rath *et al.* [50] combined process, stakeholder, structural, and textual similarity metrics. Mazrae *et al.* [37] incorporated non-textual cues such as authorship, timestamps, and status. Dong *et al.* [15], a semi-supervised framework, tackled data imbalance and sparsity. These approaches were later improved by deep learning.

Deep Learning-Based Approaches: Recent works focused on the use of deep learning. Ruan *et al.* [55] learned semantic representations with word embeddings and RNNs, while Xie *et al.* [70] combined RNNs with SVMs and a code knowledge graph from ASTs for semantic and code context. Lin *et al.* [32] leveraged a BERT-based framework pre-trained on CodeSearchNet and fine-tuned on small datasets to address data sparsity. Zhu *et al.* [73] employed deep semi-supervised learning and iteratively retrained its model using pseudo-labels on unlabeled data. Zhang *et al.* [71] employed knowledge distillation to improve both accuracy and efficiency. Despite these improvements, we found that the approaches were assessed using evaluations whose realism could be improved.

Replication Studies: Our study found that the data used in evaluations for issue-commit link recovery may have the drawback of an unrealistic distribution of false links. Other studies [8, 14, 28, 33, 59] have emphasized the importance of using methods and data that evaluate tools in different settings. While other works [15, 32, 37, 55, 71] have investigated issues of data cleanliness and data leakage, our work is the first replication study of issue-commit linkers and highlights the importance of evaluating them using realistic evaluation data that match the real development history.

9 Conclusion and Future Work

In this study, we successfully replicate the strong performance of the state-of-the-art work on issue-commit linking. To investigate it further, we constructed a new benchmark under a proposed Realistic Distribution Setting (RDS) that adaptively constructs false links based on the level of development activity in the same time frame as the ground-truth link. To the best of our knowledge, the benchmark is the largest in the literature, consisting of 9,319 unique issues from 20 open-source projects, with an average of 1,530 false links constructed per issue. We find that the use of an off-the-shelf IR method outperforms the state-of-the-art technique. Building on our findings, we propose **EasyLink**, a scalable and efficient approach that combines retrieval with an additional step of reranking using an LLM. In terms of average Precision@1, EasyLink outperforms the state-of-the-art approach by more than 4 times.

In the future, we plan to extend EasyLink for other tasks in software traceability, such as mapping features to their implementations and linking requirements to code changes.

10 Data Availability

The implementation and documentation of EasyLink, along with our collected evaluation dataset, have been anonymized and made available at this URL link

https://figshare.com/s/d495f11c4cc5c1c72e68

References

[1] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. 2002. Recovering traceability links between code and documentation. *IEEE transactions on software engineering* 28, 10 (2002), 970–983.

[2] Thazin Win Win Aung, Huan Huo, and Yulei Sui. 2020. A literature review of automatic traceability links recovery for software change impact analysis. In *Proceedings of the 28th International Conference on Program Comprehension*. 14–24.

[3] Adrian Bachmann and Abraham Bernstein. 2009. 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*. 119–128.

[4] Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. 2010. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. 97–106.

[5] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. 2009. Fair and balanced? bias in bug-fix datasets. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 121–130.

[6] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[7] Markus Borg, Per Runeson, and Anders Ardö. 2014. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering* 19 (2014), 1565–1616.

[8] Partha Chakraborty, Krishna Kanth Arumugam, Mahmoud Alfadel, Meiyappan Nagappan, and Shane McIntosh. 2024. Revisiting the performance of deep learning-based vulnerability detection on realistic datasets. *IEEE Transactions on Software Engineering* (2024).

[9] Harrison Chase. 2022. LangChain. https://github.com/langchain-ai/langchain Accessed: 2025-02-10.

[10] Jane Cleland-Huang, Orlena Gotel, Andrea Zisman, et al. 2012. *Software and systems traceability*. Vol. 2. Springer.

[11] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. 2014. Software traceability: trends and future directions. In *Future of Software Engineering Proceedings* (Hyderabad, India) *(FOSE 2014)*. Association for Computing Machinery, New York, NY, USA, 55–69. doi:10.1145/2593882.2593891

[12] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. routledge.

[13] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[14] Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. 2024. Vulnerability detection with code language models: How far are we? *arXiv preprint arXiv:2403.18624* (2024).

[15] Liming Dong, He Zhang, Wei Liu, Zhiluo Weng, and Hongyu Kuang. 2022. Semi-supervised pre-processing for learning-based traceability framework on real-world software projects. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 570–582.

[16] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[17] Marc Eaddy, Alfred V Aho, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. 2008. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *2008 16th IEEE International Conference on Program Comprehension*. Ieee, 53–62.

[18] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[19] Wei Fu and Tim Menzies. 2017. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 49–60.

[20] Dominik Fuchß, Tobias Hey, Jan Keim, Haoyu Liu, Niklas Ewald, Tobias Thirolf, and Anne Koziolek. 2025. LiSSA: Toward Generic Traceability Link Recovery through Retrieval-Augmented Generation. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering. ICSE*, Vol. 25.

[21] Jingtong Gao, Bo Chen, Xiangyu Zhao, Weiwen Liu, Xiangyang Li, Yichao Wang, Zijian Zhang, Wanyu Wang, Yuyang Ye, Shanru Lin, et al. 2024. LLM-enhanced Reranking in Recommender Systems. *arXiv preprint arXiv:2406.12433* (2024).

[22] Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. 2011. On integrating orthogonal information retrieval methods to improve traceability recovery. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 133–142.

[23] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 3–14.

[24] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. 2006. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering* 32, 1 (2006), 4–19.

[25] Abram Hindle, Daniel M German, and Ric Holt. 2008. What do large commits tell us? a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*. 99–108.

[26] Raisa Islam and Owana Marzia Moushi. 2024. Gpt-4o: The cutting-edge advancement in multimodal llm. *Authorea Preprints* (2024).

[27] Maliheh Izadi, Pooya Rostami Mazrae, Tom Mens, and Arie van Deursen. 2022. Linkformer: Automatic contextualised link recovery of software artifacts in both project-based and transfer learning settings. *arXiv preprint* (2022).

[28] Hong Jin Kang, Khai Loong Aw, and David Lo. 2022. Detecting false alarms from automatic static analysis tools: How far are we?. In *Proceedings of the 44th International Conference on Software Engineering*. 698–709.

[29] Masanari Kondo, Yutaro Kashiwa, Yasutaka Kamei, and Osamu Mizuno. 2022. An empirical study of issue-link algorithms: which issue-link algorithms should we use? *Empirical Software Engineering* 27, 6 (2022), 136.

[30] Tien-Duy B Le, Mario Linares-Vásquez, David Lo, and Denys Poshyvanyk. 2015. Rclinker: Automated linking of issue reports and commits leveraging rich contextual information. In *2015 IEEE 23rd international conference on program comprehension*. IEEE, 36–47.

[31] Kaixuan Li, Jian Zhang, Sen Chen, Han Liu, Yang Liu, and Yixiang Chen. 2024. PatchFinder: A two-phase approach to security patch tracing for disclosed vulnerabilities in open-source software. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 590–602.

[32] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability transformed: Generating more accurate links with pre-trained bert models. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). *IEEE, 324ś335* (2021).

[33] Yue Liu, Chakkrit Tantithamthavorn, Yonghui Liu, Patanamon Thongtanunam, and Li Li. 2025. Automatically Recommend Code Updates: Are We There Yet? *ACM Transactions on Software Engineering and Methodology* 33, 8 (2025), 1–27.

[34] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 16, 4 (2007), 13–es.

[35] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[36] Andrian Marcus, Jonathan I Maletic, and Andrey Sergeyev. 2005. Recovery of traceability links between software documentation and source code. *International Journal of Software Engineering and Knowledge Engineering* 15, 05 (2005), 811–836.

[37] Pooya Rostami Mazrae, Maliheh Izadi, and Abbas Heydarnoori. 2021. Automated recovery of issue-commit links leveraging both textual and non-textual data. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 263–273.

[38] Andrew Meneely, Harshavardhan Srinivasan, Ayemi Musa, Alberto Rodriguez Tejeda, Matthew Mokary, and Brian Spates. 2013. When a patch goes bad: Exploring the properties of vulnerability-contributing commits. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 65–74.

[39] Leila Naslavsky and Debra J Richardson. 2007. Using traceability to support model-based regression testing. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*. 567–570.

[40] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. 2022. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005* (2022).

[41] Anh Tuan Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2012. 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*. 1–11.

[42] Truong Giang Nguyen, Thanh Le-Cong, Hong Jin Kang, Xuan-Bach D Le, and David Lo. 2022. Vulcurator: a vulnerability-fixing commit detector. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1726–1730.

[43] Giang Nguyen-Truong, Hong Jin Kang, David Lo, Abhishek Sharma, Andrew E Santosa, Asankhaya Sharma, and Ming Yi Ang. 2022. Hermes: Using commit-issue linking to detect vulnerability-fixing commits. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 51–62.

[44] Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. 2010. On the equivalence of information retrieval methods for automated traceability link recovery. In *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 68–71.

[45] OpenAI. 2024. GPT-4o. https://openai.com/index/hello-gpt-4o/.
[46] EasyLink: Replication Package. 2025. Replication package of EasyLink. https://figshare.com/s/d495f11c4cc5c1c72e68.
[47] Michael C Panis. 2010. Successful deployment of requirements traceability in a commercial engineering organization... really. In *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 303–307.
[48] Ranjith Purushothaman and Dewayne E Perry. 2005. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering* 31, 6 (2005), 511–526.
[49] Filip Radlinski and Nick Craswell. 2010. Comparing the sensitivity of information retrieval metrics. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 667–674.
[50] Michael Rath, Jacob Rendall, Jin LC Guo, Jane Cleland-Huang, and Patrick Mäder. 2018. Traceability in the wild: automatically augmenting incomplete trace links. In *Proceedings of the 40th International Conference on Software Engineering*. 834–845.
[51] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3982–3992.
[52] Irving Muller Rodrigues, Daniel Aloise, Eraldo Rezende Fernandes, and Michel Dagenais. 2020. A soft alignment model for bug deduplication. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 43–53.
[53] Alberto D Rodriguez, Katherine R Dearstyne, and Jane Cleland-Huang. 2023. Prompts matter: Insights and strategies for prompt engineering in automated software traceability. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 455–464.
[54] Bilyaminu Auwal Romo, Andrea Capiluppi, and Tracy Hall. 2014. Filling the Gaps of Development Logs and Bug Issue Data. In *Proceedings of The International Symposium on Open Collaboration* (Berlin, Germany) *(OpenSym '14)*. Association for Computing Machinery, New York, NY, USA, 1–4. doi:10.1145/2641580.2641592
[55] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *Journal of Systems and Software* 158 (2019), 110406.
[56] Tetsuya Sakai and Noriko Kando. 2008. On information retrieval metrics designed for evaluation with incomplete relevance assessments. *Information Retrieval* 11 (2008), 447–470.
[57] Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (1975), 613–620.
[58] Gerald Schermann, Martin Brandtner, Sebastiano Panichella, Philipp Leitner, and Harald Gall. 2015. Discovering loners and phantoms in commit and issue data. In *2015 IEEE 23rd International conference on program comprehension*. IEEE, 4–14.
[59] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on software engineering* 39, 9 (2013), 1208–1215.
[60] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems* 33 (2020), 16857–16867.
[61] George Spanoudakis and Andrea Zisman. 2005. Software traceability: a roadmap. In *Handbook of software engineering and knowledge engineering: vol 3: recent advances*. World Scientific, 395–428.
[62] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. *arXiv preprint arXiv:2304.09542* (2023).
[63] Yan Sun, Celia Chen, Qing Wang, and Barry Boehm. 2017. Improving missing issue-commit link recovery using positive and unlabeled data. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 147–152.
[64] Yan Sun, Qing Wang, and Ye Yang. 2017. Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance. *Information and Software Technology* 84 (2017), 33–47.
[65] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.
[66] Tianyu Wang, Nianjun Zhou, and Zhixiong Chen. 2024. Enhancing computer programming education with llms: A study on effective prompt engineering for python code generation. *arXiv preprint arXiv:2407.05437* (2024).
[67] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems* 33 (2020), 5776–5788.
[68] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 15–25.
[69] Zhaonan Wu, Yanjie Zhao, Chen Wei, Zirui Wan, Yue Liu, and Haoyu Wang. 2025. CommitShield: Tracking Vulnerability Introduction and Fix in Version Control Systems. *arXiv preprint arXiv:2501.03626* (2025).
[70] Rui Xie, Long Chen, Wei Ye, Zhiyu Li, Tianxiang Hu, Dongdong Du, and Shikun Zhang. 2019. Deeplink: A code knowledge graph based deep learning approach for issue-commit link recovery. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 434–444.
[71] Chenyuan Zhang, Yanlin Wang, Zhao Wei, Yong Xu, Juhong Wang, Hui Li, and Rongrong Ji. 2023. EALink: An efficient and accurate pre-trained framework for issue-commit link recovery. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 217–229.
[72] Ting Zhang, DongGyun Han, Venkatesh Vinayakarao, Ivana Clairine Irsan, Bowen Xu, Ferdian Thung, David Lo, and Lingxiao Jiang. 2023. Duplicate bug report detection: How far are we? *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–32.
[73] Jianfei Zhu, Guanping Xiao, Zheng Zheng, and Yulei Sui. 2024. Deep semi-supervised learning for recovering traceability links between issues and commits. *Journal of Systems and Software* (2024), 112109.
[74] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107* (2023).