LinkRank: A Learning-to-Rank Framework for One-to-Many Issue–Commit Traceability

Abhishek Kumar, Tuhin Mondal, Partha Pratim Das, and Partha Pratim Chakrabarti

***Abstract*—Recovering traceability links between issues and commits is a fundamental requirement for effective software maintenance, comprehension, and analytics. Yet, most existing research has restricted the problem to one-to-one mappings, overlooking the common one-to-many scenarios where a single issue is resolved through multiple commits. This simplification limits the realism and applicability of current models in large-scale development. To address this gap, we propose LinkRank, a learning-to-rank framework that formulates one-to-many issue–commit recovery as a ranking problem. LinkRank integrates lightweight lexical and retrieval-based representations with a LambdaMART ranker and employs an iterative selection mechanism that operates under both oracle (Known-K) and realistic (Unknown-K) stopping strategies. We further extend this approach with LinkRank-C2I, a complementary commit-to-issue variant that reinforces bidirectional consistency. To enable systematic evaluation, we construct a novel GitHub dataset that explicitly captures realistic one-to-many relationships between issues and commits. Extensive experiments across multiple repositories demonstrate that LinkRank and LinkRank-C2I substantially outperform existing baselines in both precision and F1-score while remaining computationally efficient. Our findings establish learning-to-rank as a robust and scalable paradigm for advancing practical issue–commit traceability.**

***Index Terms*—Issue-Commit Linking, Software Traceability, Repository Mining, Empirical Software Engineering & Trace Link Recovery.**

I. Introduction

Software traceability refers to the process of establishing and maintaining connections between various software artifacts, helping developers understand the relationships between different components of a software system [1], [2]. It plays a crucial role in understanding the impact of software changes [3], bug fixing, safety systems [4], and project management [5]. Among the various aspects of traceability, one of the most significant is issue-commit linking, which establishes connections between reported issues and the commits made to address them [6]–[8].

Issues are typically handled in issue tracking systems such as *BugZilla* [9] and *Backlog* [10], while commits are recorded and updated in version control systems like *Git* [11] and *Mercurial* [12], with these systems often operating independently. Developers can tag commits with corresponding issues, but this practice is often inconsistent. As a result, gaps frequently arise in linking issues to their respective commits [13], leading to increased software maintenance costs [14]–[16]. Establishing these connections is crucial for various research areas, including bug prediction [17] and commit analysis [18]. These links are essential for understanding why specific code changes were introduced, helping developers maintain, update, and refactor software more efficiently.

To address this, researchers have proposed various automated approaches over the years. Early methods primarily relied on heuristic-based techniques, such as feature engineering and manual rule implementation [17], [19], [20]. However, these approaches struggled with low precision and poor generalizability across different projects. To address these limitations, subsequent works introduced machine learning techniques [21], [22], [23], [24], [25], which automated the learning process from textual and metadata features. More recently, deep learning [26], [27] and pre-trained transformer models [7], [28]–[31] have been employed to capture semantic relationships between issues and commits by using neural networks trained on large datasets.

While existing research has significantly advanced issue-commit link recovery, as per our knowledge, it has focused solely on one-to-one relationships, where a single issue is linked to a single commit. However, real-world software development often follows a more complex pattern, where a single issue is addressed through multiple commits, forming a one-to-many relationship [7], [17]. As an example, Figure 1 shows a GitHub issue linked to several commits in the same pull request. Such cases also arise in large feature implementations, bug fixes requiring multiple iterations, and incremental enhancements in agile development. Ignoring these one-to-many links reduces the effectiveness of traceability models, as they fail to capture the full extent of an issue's resolution. This gap also impacts debugging, maintenance, quality assurance, and historical analysis [6].

To fill this gap, we present *LinkRank*, a learning-to-rank formulation designed for one-to-many issue-commit linking. Each issue acts as the query, and candidate commits are scored using a compact blend of lexical signals (TF–IDF + SVD) and retrieval focus (BM25), with ranking performed by a LambdaMART model. Selection proceeds via an *iterative pick–remove–renormalize* loop that supports both Known-K and Unknown-K regimes, where K denotes the true number of commits associated with an issue. We also study a complementary commit→issue variant, *LinkRank-C2I*, that performs bidirectional refinement. While we experiment with incorporating semantic embeddings such as CodeBERT, our results show that the core algorithm itself remains highly effective, demonstrating that LinkRank does not depend on heavy transformer models. Our evaluation shows that, LinkRank achieves an average F-score of 85.98%, while its variant, LinkRank-C2I, attains 83.44%. These results are substantially higher than

A. Kumar, T. Mondal and P. P. Chakrabarti are with Indian Institute of Technology Kharagpur, India (e-mails: abhishek16@kgpian.iitkgp.ac.in, tuhin@kgpian.iitkgp.ac.in, ppc@iitkgp.ac.in).

P. P. Das is with Ashoka University, India (e-mail: partha.das@ashoka.edu.in).

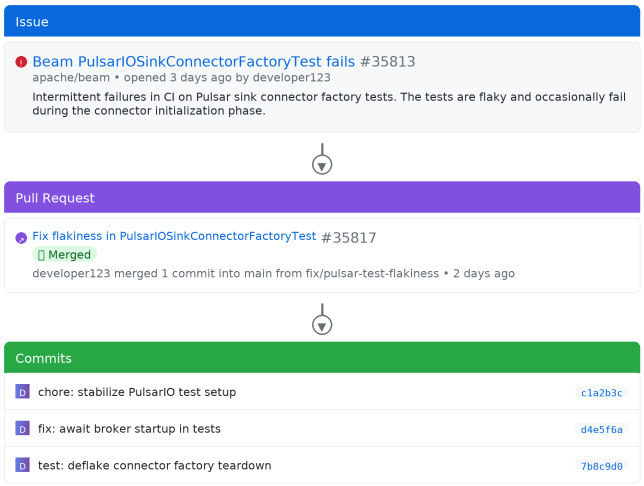

Fig. 1. This example demonstrates how a single issue (#35813) is resolved through a pull request (#35817) that bundles multiple focused commits. Each commit addresses a specific aspect of the problem: test setup stabilization, broker startup synchronization, and teardown process improvement.

those of state-of-the-art baselines such as EALink (61.47%), HybridLinker (58.90%), FRLink (55.84%), and DeepLink (48.51%), underscoring the effectiveness and robustness of our learning-to-rank formulation for one-to-many issue–commit link recovery.

In summary, this paper makes the following key contributions:

1) *One-to-many dataset.* We construct a new dataset by mining GitHub pull requests (PRs) that are linked to *exactly one* issue and contain *two to six* commits, resulting in genuine one-to-many relations while avoiding degenerate or outlier cases. This careful filtering, together with repository-aware negative/false link construction, is crucial for realistic evaluation and reduces ambiguity from multi-issue PRs.
2) *LinkRank framework.* We cast linking as an issue-centric ranking task and learn a LambdaMART scorer over pairwise features: lexical similarity (TF–IDF+SVD) and retrieval focus (BM25). At inference, LinkRank *picks* the top-scoring commit for an issue, *removes* it, *renormalizes* scores within the remaining pool, and *repeats*, with stopping via *Known-K* (top-K) or *Unknown-K* (ABS/REL thresholds).
3) *Optional semantic embeddings.* To test robustness, we add CodeBERT-based semantic similarity as an additional feature channel. The marginal gains observed confirm that LinkRank's strength derives primarily from its ranking formulation and IR-style features, making it efficient and not reliant on transformers.
4) *LinkRank-C2I variant.* We further experiment with a bidirectional refinement pipeline: first shortlist issues per commit (commit→issue ranking), then validate from the issue side (issue→commit re-ranking) using the same iterative selection policy. This cross-check improves precision while preserving recall and complements the primary formulation.
5) *Issue-wise (macro) evaluation protocol.* For one-to-many linking, we evaluate *per issue* by comparing the predicted set $\widehat{\mathcal{C}}(i)$ to the gold set $\mathcal{C}^{\star}(i)$ using set-based Precision, Recall, and F1, and report *macro averages* across issues. This directly measures completeness and avoids the optimism of pairwise link-level scoring.

The remainder of this paper is structured as follows. Section II reviews the related work. Section III introduces the proposed LINKRANK framework and its variant LINKRANK-C2I, detailing the feature design, LambdaMART training, and selection strategies. Section IV describes the experimental setup and presents the results across multiple repositories. Section V discusses key findings, limitations, and directions for future work. Finally, Section VI concludes the paper.

II. RELATED WORK

Rule Based Approach: Bird et al. [32] observed that developers often struggle with manually identifying missing links between issues and commits, a process that is both time-consuming and tedious. To assist developers in this task, they introduced LINKSTER, a system that provides query interfaces for both issue and commit data, making it easier to retrieve relevant links. However, LINKSTER still required manual effort, as it did not fully automate the link recovery process. To overcome this limitation, Wu et al. developed ReLink [17], the first automatic approach for recovering missing links between issues and commits. ReLink leveraged textual similarity by treating issue reports and commit log messages as plain text, allowing for automated text feature extraction and comparison. While this marked a significant step forward, the reliance solely on textual similarity proved insufficient, as it failed to capture cases where issue descriptions and commit messages were semantically related but lexically different. Recognizing this shortcoming, Nguyen et al. [19] introduced MLINK, an approach that extended ReLink by incorporating both textual features and structural information from the modified source code. Unlike ReLink, which focused exclusively on textual similarity, MLINK considered actual code changes associated with bug fixes, enabling a more comprehensive and accurate recovery of missing links. By integrating both natural language processing techniques and software-specific modifications, MLINK significantly improved the precision of issue-commit link recovery.

Machine Learning Approach: Previous approaches, such as ReLink and MLink, relied on textual similarities between commit logs and issue reports, often failing when commit messages were vague or missing. RCLinker [24] addressed this by integrating ChangeScribe [33], [34], which generates commit messages summarizing code changes. By combining these generated summaries with developers' original messages, RCLinker enriched textual features and adopted a classification-based approach using a random forest classifier to predict issue-commit links. While RCLinker improved upon prior methods, it still overlooked non-source documents present in commits. FRLink [25] tackled this limitation by incorporating such documents while filtering out irrelevant files, enhancing contextual information beyond conventional

text-based approaches. However, most existing models framed the problem as binary classification, disregarding unlabelled links. Recognizing this, PULink [23] reframed link recovery as a positive-unlabelled (PU) learning task, distinguishing unlabelled links from negative ones, leading to better model training and improved performance. Despite these advancements, prior models still suffered from low precision and high computational costs, particularly due to the scarcity of textual data. HybridLinker [21] addressed this by combining textual and non-textual information in a hybrid model. It trained separate classifiers for both data types and fused their outputs using a weighted ensemble technique, resulting in higher precision, lower computational cost, and improved link recovery accuracy.

Deep Learning Approach: DeepLink [26], [27] stands out as one of the first deep learning models designed to tackle the issue-commit link recovery problem. Unlike previous approaches that extract text and code features from commit logs and issue reports, resulting in the loss of semantic information, DeepLink focuses on capturing the crucial semantic details that can improve the learning process for automatic issue-commit link recovery. To achieve this, DeepLink constructs a code knowledge graph that captures the semantic meaning of the code, setting it apart from earlier methods that primarily rely on textual and code similarity.

Pre-trained Transformers: Pre-trained transformer-based models have significantly advanced issue-commit link recovery by using large-scale language modeling to capture deep semantic relationships between issue descriptions and commit messages. T-BERT [28] introduced a three-step transfer learning approach, addressing data scarcity and computational inefficiency while demonstrating superior accuracy over traditional models. BTLink [29] further improved link recovery by utilizing dual BERT encoders to process both issue text and commit data, integrating semantic representations through a fusion layer, and outperforming state-of-the-art approaches in precision, recall, and cross-project adaptability. EALink [7] refined transformer-based methods by tackling computational inefficiency, inter-commit correlations, and multi-purpose commit handling using knowledge distillation and contrastive learning, achieving substantial performance gains while reducing computational costs. Although their dataset includes certain issues linked to multiple commits, the exact distribution of commits per issue is not reported, and their formulation remains inherently one-to-one, treating each issue–commit pair as an independent binary classification task.

III. The LinkRank Approach

Our approach is organized into four phases: (1) Dataset construction process, (2) Feature Representations, (3) Learning-to-Rank with LambdaMART, and (4) Selection Strategies. The complete workflow is illustrated in Figure 2, and each phase is discussed in the following subsections.

A. Phase I: Dataset construction process

The dataset for this study was constructed using GitHub's API to extract issue–commit pairs from multiple repositories. GitHub imposes a rate limit of 5,000 requests per hour per authenticated user, making data extraction a time-consuming process [35]. To ensure that the dataset captured realistic one-to-many relationships, we specifically filtered issues that were linked to at least two and at most six commits. Identifying repositories that contained a sufficiently large number of such issues proved challenging, as many projects either had too few one-to-many cases or exhibited extreme imbalance. Therefore, we selected repositories with a sufficient number of issues satisfying the 2–6 commit criterion. The overall dataset statistics are summarized in Table I, providing a detailed view of the scale and distribution of the collected data. This systematic filtering process ensured both the scalability of the dataset and its fidelity to real-world development practices.

1) False Link Generation: Constructing a reliable set of false links is critical for evaluating issue-commit link recovery models, as it enables a more accurate assessment of the models' ability to distinguish between correctly linked and mismatched issue-commit pairs. However, previous works have often faced challenges in generating false links, which may compromise the quality of the datasets used for model evaluation.

Several existing studies have relied on simplistic strategies for generating false links, which can lead to inaccuracies. For example, BTLink [29] and DeepLink [27] generated false links by randomly pairing issues and commits from separate projects or unrelated time periods. This approach risks introducing semantically related links that were mistakenly labeled as false, as code changes made in close time proximity or in related repositories may still be connected to the same issue. Similarly, HERMES [6] and EALink [7] created false links using time-based sampling techniques, where issues and commits were paired if they were not linked within a specific time frame. However, this method overlooks the possibility that complex issue resolutions may involve multiple commits spread over extended periods, potentially leading to the inclusion of valid links being mislabeled as false. Additionally, T-BERT [30] generated a large number of false links without verifying their semantic separation from true links, leading to datasets where negative pairs may still be contextually relevant, thus reducing the reliability of the ground truth.

To address these limitations, we adopt a Pull Request (PR)-aware strategy for generating false links. We first identify valid PRs that satisfy our condition of having between 2 and 6 commits. For each valid PR p, we denote its corresponding issue as i_p and the set of associated commits as $C_p = \{c_1, c_2, \ldots, c_m\}$, where $2 \leq m \leq 6$. The set of true links is therefore defined as:

$$T = \{(i_p, c_j) \mid p \in \mathcal{P}_{\text{valid}},\ c_j \in C_p\}.$$

Next, we consider PRs that do not satisfy the 2–6 commit constraint, denoted as $\mathcal{P}_{\text{invalid}}$. From these PRs, we construct a pool of unrelated commits:

$$C_{\text{invalid}} = \bigcup_{p \in \mathcal{P}_{\text{invalid}}} C_p.$$

For each valid issue i_p with $m = |C_p|$ true commits, we generate an equal number of false links by randomly

TABLE I
STATISTICS OF SELECTED GITHUB REPOSITORIES USED FOR DATASET CONSTRUCTION.

Project Name	Stars	Commits	Unique Issues	Number of Issues with N Commits				
				Two Commits	Three Commits	Four Commits	Five Commits	Six Commits
Apache/Beam	8.3k	3104	486	206	110	81	54	31
Apache/Datafusion	7.7k	3534	496	135	132	95	79	52
Apache/Superset	67.8k	3136	498	230	105	65	56	41
Apache/Mxnet	20.8k	1216	187	69	49	36	19	14
Apache/Dubbo	41.3k	1205	201	97	46	22	20	16
Apache/Iceberg	7.9k	1750	257	97	57	36	39	26
Kubernetes	117k	2948	483	238	102	66	43	31
grpc	43.6k	2162	486	131	84	51	34	28
TensorFlow	191k	2719	408	167	86	69	45	35
PyTorch	93.2k	748	125	51	45	14	9	6

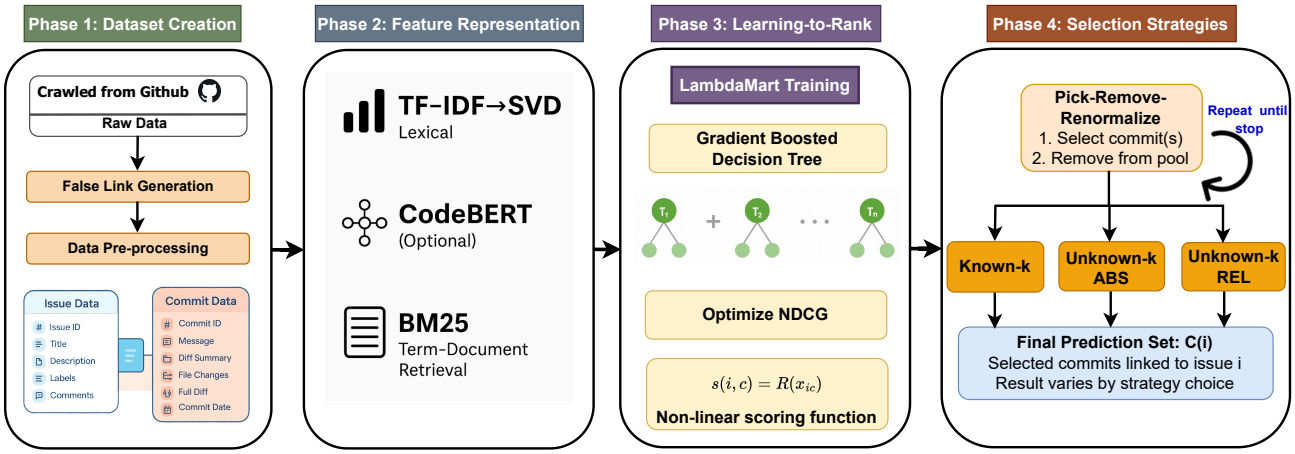

Fig. 2. Overall workflow of the proposed LINKRANK framework, illustrating its four main phases: Data preparation, Feature Representations, Learning-to-Rank, and Selection Strategies.

sampling m unrelated commits from C_{invalid}. This ensures that the number of false links matches the number of true links on a per-issue basis:

$$F_p = \{(i_p, c_r) \mid c_r \sim \text{Uniform}(C_{\text{invalid}}),\ |F_p| = m\}.$$

Finally, the dataset for each repository is formed by combining true and false links:

$$D = T \cup \bigcup_{p \in \mathcal{P}_{\text{valid}}} F_p.$$

This construction guarantees balance between positive and negative samples for each issue, while keeping the false links realistic and repository-aware.

B. Phase II: Feature Representations

To effectively model the one-to-many issue–commit linking problem, each candidate pair (i, c) is encoded into a feature vector x_{ic} that captures complementary dimensions of similarity. Rather than relying on a single view of the data, we combine lexical, retrieval-based, and optional semantic features, enabling the model to distinguish true links from superficially similar but incorrect ones.

1) Lexical Similarity (TF–IDF + SVD).: For the lexical perspective, we employ Term Frequency–Inverse Document Frequency (TF–IDF), a well-established method in information retrieval for quantifying the salience of terms relative to a corpus. The TF component reflects how often a term appears in a document (issue or commit), while the IDF component down-weights terms that are frequent across the whole corpus and up-weights rarer, more discriminative terms. To capture latent topics and reduce dimensionality, we apply truncated Singular Value Decomposition (SVD), yielding compact dense representations. The cosine similarity between an issue vector v_i and a commit vector v_c is then

$$f_{\text{text}}(i, c) = \frac{\langle v_i, v_c \rangle}{\|v_i\| \cdot \|v_c\|},$$

where $\langle v_i, v_c \rangle$ is the dot product and $\|v_i\|$, $\|v_c\|$ are vector norms. High values indicate strong lexical alignment, while the SVD projection smooths sparsity and captures latent semantics beyond exact token overlap.

2) BM25 Matching.: While TF–IDF provides global representations, retrieval often benefits from query-focused matching. BM25 [36], a probabilistic ranking function widely used in search engines, is particularly effective here. It improves

upon TF–IDF by modeling diminishing returns for repeated term occurrences and by normalizing document length, thus avoiding bias toward longer commit messages. Treating the issue text as a query and the commit text as a document, BM25 yields

$$f_{\text{bm25}}(i,c) = \sum_{t \in i} \text{IDF}(t) \cdot \frac{f(t,c) \cdot (k_1+1)}{f(t,c) + k_1 \cdot \left(1 - b + b \cdot \frac{|c|}{\text{avgdl}}\right)}.$$

Here, $\text{IDF}(t)$ emphasizes rare terms, $f(t,c)$ counts term occurrences in commit c, $|c|$ is the commit length, avgdl is the average commit length in the *training* corpus, k_1 controls frequency saturation, and b tunes length normalization. In this way, BM25 captures how well a commit matches an issue when the issue is interpreted as a retrieval query.

3) Semantic Similarity (CodeBERT).: In addition to lexical and retrieval features, we include a complementary semantic signal from CodeBERT [37], a transformer model pre-trained on natural language and code. We compute mean-pooled embeddings $\phi(i)$ and $\phi(c)$ for the issue and commit, L2-normalize them, and score with cosine similarity:

$$f_{\text{sem}}(i,c) = \cos\big(\phi(i), \phi(c)\big).$$

This semantic channel complements TF–IDF and BM25 by capturing paraphrases and code-aware context (e.g., terse messages or alternate phrasings). Importantly, our ablations show that the model remains strong *without* CodeBERT; adding it primarily improves robustness in cases where wording diverges or context is sparse.

C. Phase III: Learning-to-Rank with LambdaMART

To transform the extracted feature representations into an effective linking model, we employ *LambdaMART* [38], a gradient boosted decision tree algorithm specifically designed for ranking tasks. Unlike conventional classification methods that predict binary labels, LambdaMART directly optimizes ranking-based metrics such as the Normalized Discounted Cumulative Gain (NDCG), making it particularly well suited for the one-to-many issue–commit linking problem.

The central idea is to model relative preferences between commits for each issue. Training data are grouped by issue, ensuring that the model does not learn absolute scores across unrelated issues, but instead learns how to order candidate commits within each issue context (Algorithm 1). Given the feature vector x_{ic} for an issue–commit pair (i,c), LambdaMART learns a non-linear scoring function $\mathcal{R}$ through gradient-boosted regression trees. The output of the model is a ranking score:

$$s(i,c) = \mathcal{R}(x_{ic}),$$

where higher scores indicate stronger likelihood of a true issue–commit link.

This formulation offers two key benefits. First, it allows the model to focus directly on ranking quality rather than classification accuracy, which is crucial in settings where each issue may be linked to multiple commits. Second, tree-based boosting naturally captures complex, non-linear feature interactions (e.g., when lexical similarity is high but BM25 matching is weak, or when semantic similarity diverges), yielding a more discriminative and flexible ranking function. At inference time, we score a pool of candidate commits for each issue and then apply an *iterative* selection rule: we pick the current top commit, remove it from the pool, recompute per-issue normalization $\tilde{s}(i,c)$ (min–max scaled) and the updated $s_{\max}$, and repeat until the stopping rule is met. Thus, LambdaMART forms the foundation of our LINKRANK framework, enabling robust recovery of one-to-many issue–commit links.

D. Phase IV: Selection Strategies

Once the ranking scores are obtained, the final prediction set $\widehat{\mathcal{C}}(i)$ for each issue i must be constructed. We follow an *iterative pick–remove–renormalize* procedure: at each step, select according to the chosen policy, remove the selected commit from the pool, and recompute per-issue normalization (min–max $\tilde{s}$) and the current $s_{\max}$ before the next step. We distinguish between two cases:

a) Known-K: In this case, we assume that the true number of commits K associated with issue i is known (an oracle setting). We iteratively take the top-ranked commit, remove it, renormalize scores, and continue until exactly K commits have been selected. Although impractical in deployment (since the true K is rarely available), this strategy serves as an important upper bound to evaluate the performance of our framework.

b) Unknown-K: In this case, the number of commits linked to each issue is not known beforehand. This makes the task considerably harder, as the system must automatically determine not only which commits to link, but also how many. To address this challenge, we introduce two threshold-based strategies that adapt selection to the relative or absolute quality of ranking scores:

- *Absolute Thresholding (ABS):* At each iteration, after renormalization, link any commit whose normalized score $\tilde{s}(i,c)$ exceeds a fixed threshold τ; remove selected commits and repeat until no remaining candidate exceeds τ. This imposes a global cutoff across issues.
- *Relative Thresholding (REL):* At each iteration, with the current top score $s_{\max}(i)$, link any commit satisfying $s(i,c) \geq \gamma \cdot s_{\max}(i)$; remove selected commits, recompute $s_{\max}(i)$, and continue until no remaining candidate satisfies the γ criterion. This adapts to per-issue score scales.

Together, these strategies allow us to contrast oracle-based performance (Known-K) with more realistic scenarios (Unknown-K). The ABS rule emphasizes global calibration, while the REL rule emphasizes local adaptivity, and both operate within the same iterative pick–remove–renormalize loop, making them well-suited for the one-to-many linking problem.

E. Variant: LinkRank-C2I

In addition to our primary framework, LINKRANK, which adopts an issue-centric perspective, we also experimented with a variant called LINKRANK-C2I (commit→issue). The

Algorithm 1 LinkRank

Input: Issues $\mathcal{I}$, commits $\mathcal{C}$, labeled pairs $\mathcal{D}$
Output: Predicted links $\{\widehat{\mathcal{C}}(i)\}_{i\in\mathcal{I}}$
Build corpora $S_{\mathcal{I}}, S_{\mathcal{C}}$; compute TF–IDF+SVD and BM25 features; *(optional)* CodeBERT features
for all $(i, c, y) \in \mathcal{D}$ **do**
  Build feature $x_{i,c}$
end for
Train LambdaMART ranker $\mathcal{R}$ grouping by **Issue ID**
for all $i \in \mathcal{I}$ **do**
  for all $c \in \mathcal{C}$ **do**
    $s(i, c) \leftarrow \mathcal{R}(x_{i,c})$
  end for
  Compute per-issue normalized scores $\tilde{s}(i, c)$ via min–max; let $s_{\max}(i) \leftarrow \max_c s(i, c)$
  Iterative selection: repeat
    Pick $c^\star \in \arg\max_c s(i, c)$ if admissible by the chosen policy
    Add $c^\star$ to $\widehat{\mathcal{C}}(i)$; remove $c^\star$ from the candidate pool
    Recompute $\tilde{s}(i, c)$ (min–max) and update $s_{\max}(i)$
  until stopping criterion is met
  Selection policies:
    Known-K: select exactly the top K commits
    Unknown-K (ABS): select all c with $\tilde{s}(i, c) \geq \tau$
    Unknown-K (REL): select all c with $s(i, c) \geq \gamma \cdot s_{\max}(i)$
end for

motivation is to provide a complementary view by reversing the linking perspective. Since issue–commit relationships are inherently bidirectional, examining the task from the commit side lets us study whether different linking dynamics emerge when commits are treated as queries, and it further enables a consistency check between both directions, as detailed in Algorithm 2.

Step 1: Commit-to-Issue Retrieval. Each commit $c \in \mathcal{C}$ is treated as a query and issues $i \in \mathcal{I}$ are candidates. Feature representations (TF–IDF+SVD, BM25, and optional CodeBERT) are computed exactly as in LinkRank, but tuples are grouped by **commit** for training. A LambdaMART ranker $\mathcal{R}_{A2}$ learns to score candidate issues:

$$s_{A2}(c, i) = \mathcal{R}_{A2}(x_{c,i}),$$

where $x_{c,i}$ is the feature vector for commit c and issue i. To restrict the search space, we retain only the top-K issues per commit:

$$\mathcal{S}(c) = \text{Top-}K\{\, i \in \mathcal{I} \mid s_{A2}(c, i) \,\}.$$

(Optionally, a BM25 guard can expand the candidate set with a few high-recall issues per commit.)

Step 2: Issue-to-Commit Validation. We then reintroduce the issue-centric view of LinkRank. For each issue i, we restrict its candidate pool to those commits that shortlisted i in Step 1:

$$\mathcal{P}(i) = \{\, c \in \mathcal{C} \mid i \in \mathcal{S}(c) \,\}.$$

The Issue-to-Commit ranker $\mathcal{R}_{A1}$ (trained by grouping tuples **by issue**) is applied to this reduced pool, producing refined scores:

$$s(i, c) = \mathcal{R}_{A1}(x_{i,c}), \quad c \in \mathcal{P}(i).$$

This cross-directional gating enforces agreement between the commit→issue and issue→commit views.

Selection Strategies. We use the same policies as in LinkRank (Known-K, Unknown-K ABS, and Unknown-K REL) with the same *iterative* procedure: pick the current best commit for i, remove it from $\mathcal{P}(i)$, recompute per-issue normalization (min–max $\tilde{s}$) and the updated $s_{\max}(i)$, and continue until the stopping rule is met.

In summary, LinkRank-C2I complements the primary approach by enforcing bidirectional consistency, typically reducing false positives while preserving recall.

Algorithm 2 LinkRank-C2I

Input: Issues $\mathcal{I}$, commits $\mathcal{C}$, labeled pairs $\mathcal{D}$
Output: Predicted links $\{\widehat{\mathcal{C}}(i)\}_{i\in\mathcal{I}}$
Build corpora $S_{\mathcal{I}}, S_{\mathcal{C}}$; compute TF–IDF+SVD and BM25 features; *(optional)* CodeBERT features
for all $(i, c, y) \in \mathcal{D}$ **do**
  Build feature $x_{i,c}$
end for
Train commit→issue ranker $\mathcal{R}_{\text{C2I}}$ (group by **Commit ID**) and issue→commit ranker $\mathcal{R}_{\text{I2C}}$ (group by **Issue ID**)
Stage 1 — Commit→Issues (shortlist)
for all $c \in \mathcal{C}$ **do**
  $s_{\text{C2I}}(c, i) \leftarrow \mathcal{R}_{\text{C2I}}(x_{c,i})$ for all $i \in \mathcal{I}$
  shortlist $\mathcal{S}(c) \leftarrow$ top-K issues ranked by s_{C2I}
end for
Stage 2 — Issue→Commits (final selection)
for all $i \in \mathcal{I}$ **do**
  pool $\mathcal{P}(i) \leftarrow \{\, c \in \mathcal{C} : i \in \mathcal{S}(c) \,\}$
  $s(i, c) \leftarrow \mathcal{R}_{\text{I2C}}(x_{i,c})$ for $c \in \mathcal{P}(i)$
  sort by s; let $s_{\max}$ be top; compute per-issue $\tilde{s}$ (min–max)
  Iterative rule: after each pick, remove it from $\mathcal{P}(i)$ and recompute per-issue normalization
  Selection strategies: apply Known-K, Unknown-K ABS, or Unknown-K REL as in Algorithm 1
end for

IV. Experimental Setup

A. Experimental Settings

All experiments were conducted on a dedicated server equipped with an *NVIDIA RTX 4500 Ada Generation GPU* with *24 GB* of VRAM, running CUDA version 12.5 and driver version *555.42.06*. The system further consisted of a multi-core Intel CPU and 64 GB of RAM, running Ubuntu Linux (64-bit). Our implementation is based on Python 3.10 with `scikit-learn` and `xgboost` for learning-to-rank, and HuggingFace `transformers` for optional CodeBERT embeddings. To ensure reliable evaluation, we adopted a 5-fold stratified cross-validation strategy [39]. For each repository,

the labeled data was divided into five folds, with four folds used for training and the remaining fold reserved for testing. All reported results represent the average performance across the five folds, providing a balanced and consistent protocol for assessing model effectiveness across projects.

B. Evaluation Metrics

In a one-to-many relationship, an issue may be associated with multiple commits. Therefore, evaluating each issue–commit pair independently does not reflect the model's effectiveness in recovering complete commit sets for issues. To address this, we adopt an issue-wise evaluation strategy, where each *Issue_ID* (representing an issue with its linked commits) is treated as a unit. For every issue i, we compute the confusion matrix components: True Positives (TP_i), True Negatives (TN_i), False Positives (FP_i), and False Negatives (FN_i). These values yield Precision, Recall, and F1-score per issue, and we report their averages across all issues. This corresponds to *macro-averaging*, ensuring that each issue contributes equally regardless of its number of commits.

a) Precision.: Precision measures the proportion of correctly predicted commits among all commits predicted to be linked to an issue:

$$\text{Precision}^{(i)} = \frac{TP_i}{TP_i + FP_i}.$$

The overall precision is computed as the average across all issues:

$$\text{Precision} = \frac{1}{|I|} \sum_{i \in I} \text{Precision}^{(i)}.$$

b) Recall.: Recall measures the ability of the model to retrieve all relevant commits for an issue:

$$\text{Recall}^{(i)} = \frac{TP_i}{TP_i + FN_i}.$$

The overall recall is the average across all issues:

$$\text{Recall} = \frac{1}{|I|} \sum_{i \in I} \text{Recall}^{(i)}.$$

c) F1-Score.: The F1-score balances precision and recall for each issue:

$$\text{F1}^{(i)} = \frac{2 \cdot \text{Precision}^{(i)} \cdot \text{Recall}^{(i)}}{\text{Precision}^{(i)} + \text{Recall}^{(i)}}.$$

The final F1-score is obtained by averaging across issues:

$$\text{F1} = \frac{1}{|I|} \sum_{i \in I} \text{F1}^{(i)}.$$

V. Results

RQ1: How do LinkRank and LinkRank-C2I compare against existing baselines?

Table II presents the average performance across ten repositories, comparing our approaches with four representative baselines: EALink, HybridLink, FRLink, and DeepLink. The results clearly demonstrate that both LinkRank and LinkRank-C2I achieve substantially higher precision, recall, and F-score than all baselines under both Known-K and Unknown-K regimes.

TABLE II
Performance (in%) comparison of LinkRank, LinkRank-C2I, and baselines. These values are averaged across all datasets.

Models		*Precision*	*Recall*	*F-score*
LinkRank	Known-K	93.05	93.05	93.05
	No-K - ABS	87.04	92.72	88.80
	No-K - REL	83.99	89.92	85.98
LinkRank - C2I	Known-K	92.11	92.11	92.11
	No-K - ABS	84.61	93.16	87.45
	No-K - REL	85.83	83.92	83.44
Baseline	EALINK [7]	57.67	71.73	61.47
	HybridLink [21]	61.70	61.70	58.90
	FRLink [25]	46.43	73.85	55.84
	DeepLink [27]	49.58	51.422	48.51

Under the oracle *Known-K* setting, LinkRank achieves an F-score of 93.05%, closely followed by LinkRank-C2I at 92.11%. These values are substantially higher than the best-performing baseline, EALink, which records only 61.47%. In the more realistic *Unknown-K* scenarios, our approaches continue to perform strongly. With ABS thresholding, LinkRank attains an F-score of 88.80% and LinkRank-C2I achieves 87.45%, while the baselines remain in the 48%–61% range. Under REL thresholding, LinkRank records 85.98% and LinkRank-C2I achieves 83.44, again well above all baselines.

Takeaway. Across evaluation regimes, both LinkRank and LinkRank-C2I consistently outperform prior baselines by large margins, often exceeding them by 25–35 points in F-score. These results demonstrate that our learning-to-rank formulation is robust, generalizable, and establishes a new state of the art for one-to-many issue–commit link recovery. The weaker performance of baseline models can be attributed to their design: as discussed in the related work, existing approaches were primarily developed for one-to-one linking and treat the problem as a binary classification task, making them not well adapted to capture the complexity of one-to-many relations.

RQ2: How effective are LinkRank and its variant LinkRank-C2I in recovering one-to-many issue–commit links?

Table III reports results for both approaches under three selection regimes. As expected, *Known-K* yields high scores across datasets, but it assumes oracle knowledge of the true number of commits per issue and is therefore not a realistic deployment setting. We thus focus our analysis on the *Unknown-K* strategies (ABS and REL), which reflect practical use.

Under Unknown-K (ABS), LinkRank achieves strong and stable performance across most repositories (e.g., Apache/Superset: F1 = 91.35%; TensorFlow: 90.85%; PyTorch: 91.39%). LinkRank-C2I remains competitive and in some cases surpasses LinkRank (e.g., Apache/Mxnet: 88.98%; Apache/Dubbo: 83.54%; Kubernetes: 90.64%). These results indicate that a global threshold can work well when per-issue score calibration is consistent, with the commit→issue shortlist occasionally providing additional gains.

Under Unknown-K (REL), LinkRank-C2I generally outperforms LinkRank across repositories. Notable F1 scores include

TABLE III
PERFORMANCE (IN %) OF LINKRANK AND LINKRANK-C2I ACROSS DATASETS UNDER THREE REGIMES: KNOWN-K, AND UNKNOWN-K WITH ABS AND REL.

Dataset	Variations	LinkRank			LinkRank-C2I		
		Precision	Recall	F-score	Precision	Recall	F-score
Apache/Beam	Known K	91.29	91.29	91.29	90.35	90.32	90.33
	Unknown-K (ABS)	84.67	89.92	86.04	83.24	90.63	85.59
	Unknown-K (REL)	78.15	86.69	81.10	81.59	79.11	78.58
Apache/Datafusion	Known K	93.01	93.01	93.01	91.06	91.06	91.06
	Unknown-K (ABS)	84.38	93.22	87.46	83.99	92.52	86.86
	Unknown-K (REL)	82.16	88.72	84.09	83.50	81.52	80.89
Apache/Superset	Known K	95.04	95.04	95.04	91.89	91.89	91.89
	Unknown-K (ABS)	90.57	93.67	91.35	82.52	91.65	85.28
	Unknown-K (REL)	87.04	90.26	87.92	86.71	83.34	83.82
Apache/Mxnet	Known K	92.01	92.01	92.01	94.31	94.31	94.31
	Unknown-K (ABS)	86.79	92.79	89.14	86.43	94.10	88.98
	Unknown-K (REL)	85.93	88.73	86.63	89.45	88.38	88.08
Apache/Dubbo	Known K	92.72	92.72	92.72	91.42	91.42	91.42
	Unknown-K (ABS)	87.31	93.15	89.23	76.00	95.42	83.54
	Unknown-K (REL)	83.17	91.09	86.11	85.90	88.18	85.79
Apache/Iceberg	Known K	94.22	94.22	94.22	94.56	94.56	94.56
	Unknown-K (ABS)	86.81	94.02	89.52	91.96	93.55	91.97
	Unknown-K (REL)	87.93	91.41	89.22	88.07	85.04	85.40
Kubernetes	Known K	89.03	89.03	89.03	93.66	93.66	93.66
	Unknown-K (ABS)	81.92	87.13	82.83	87.65	95.92	90.64
	Unknown-K (REL)	73.57	83.12	76.73	89.54	88.32	87.68
grpc	Known K	95.75	95.75	95.75	86.44	86.44	86.44
	Unknown-K (ABS)	89.24	93.47	90.23	83.68	84.99	82.57
	Unknown-K (REL)	87.06	94.46	89.74	78.19	72.00	72.68
Tensorflow	Known K	95.75	95.75	95.75	92.70	92.70	92.70
	Unknown-K (ABS)	89.84	93.75	90.85	83.36	96.86	88.59
	Unknown-K (REL)	87.95	92.59	89.40	86.07	88.31	85.91
Pytorch	Known K	91.70	91.70	91.70	94.73	94.73	94.73
	Unknown-K (ABS)	88.89	96.04	91.39	87.25	96.01	90.48
	Unknown-K (REL)	86.90	92.13	88.89	89.25	85.00	85.59

Kubernetes: 87.68%, Apache/Mxnet: 88.08%, Apache/Dubbo: 85.79%, and PyTorch: 85.59%, with LinkRank-C2I also remaining competitive on others (e.g., Apache/Superset: 83.82% vs. 87.92% for LinkRank, Apache/Iceberg: 85.40% vs. 89.22% for LinkRank). This suggests that relative, per-issue thresholding benefits from the bidirectional pipeline, where commit-side shortlist gating helps adapt thresholds more effectively to local project characteristics.

Takeaways. (i) Known-K is a helpful reference point but not deployable in practice. (ii) In realistic *Unknown-K* settings, both ABS and REL yield strong one-to-many recovery across repositories: LinkRank delivers stable, high performance under ABS, while LinkRank-C2I provides a complementary, bidirectional view that often improves results, particularly under REL, without degrading overall robustness. Together, the two formulations validate that a learning-to-rank approach is well suited to issue→commit set prediction, offering reliable accuracy across diverse projects and evaluation regimes.

RQ3: Does integrating CodeBERT embeddings improve the performance of our models?

Table IV reports the results of both LinkRank and LinkRank-C2I when augmented with CodeBERT embeddings. Overall, we do not observe consistent or substantial improvements compared to the non-embedding setup: the observed gains are small, mixed across datasets, and often fall within the range of typical variance. This suggests that the core formulation of our framework already captures most of the discriminative signal required for one-to-many issue–commit recovery. **Why are gains limited?** First, our learning-to-rank approach trains and evaluates *per issue*, focusing on the relative ordering of candidates within each pool. Us-

TABLE IV
PERFORMANCE (IN %) OF LINKRANK AND LINKRANK-C2I WITH CODEBERT EMBEDDINGS ACROSS DATASETS UNDER THREE REGIMES: KNOWN-K, AND UNKNOWN-K WITH ABS AND REL.

Dataset	Variations	LinkRank			LinkRank-C2I		
		Precision	Recall	F-score	Precision	Recall	F-score
Apache/Beam	Known K	89.03	89.03	89.03	89.48	89.46	89.47
	Unknown-K (ABS)	82.73	91.58	85.53	84.22	88.91	85.14
	Unknown-K (REL)	80.84	76.39	76.57	80.29	76.94	76.66
Apache/Datafusion	Known K	91.45	91.45	91.45	90.65	90.65	90.65
	Unknown-K (ABS)	84.35	93.37	87.69	84.78	92.32	87.43
	Unknown-K (REL)	85.72	79.46	80.62	84.86	80.14	80.49
Apache/Superset	Known K	94.48	94.48	94.48	93.29	93.29	93.29
	Unknown-K (ABS)	90.64	94.51	91.79	88.65	92.28	89.49
	Unknown-K (REL)	89.31	85.88	86.60	87.25	86.50	86.07
Apache/Mxnet	Known K	90.94	90.94	90.94	94.58	94.58	94.58
	Unknown-K (ABS)	87.26	92.57	89.19	86.33	93.59	88.70
	Unknown-K (REL)	85.76	83.55	83.46	87.98	87.34	86.49
Apache/Dubbo	Known K	93.10	93.10	93.10	90.59	90.59	90.59
	Unknown-K (ABS)	88.46	91.81	89.11	82.48	91.28	85.57
	Unknown-K (REL)	87.31	82.64	83.59	84.12	87.22	84.53
Apache/iceberg	Known K	93.16	93.16	93.16	94.61	94.61	94.61
	Unknown-K (ABS)	88.77	93.51	90.08	90.06	94.66	91.61
	Unknown-K (REL)	87.99	85.53	85.59	87.35	85.57	85.41
Kubernetes	Known K	87.80	87.80	87.80	93.36	93.36	93.36
	Unknown-K (ABS)	81.46	91.46	84.93	88.38	94.63	90.32
	Unknown-K (REL)	81.78	70.22	72.97	88.35	88.15	86.67
grpc	Known K	94.49	94.49	94.49	88.02	88.02	88.02
	Unknown-K (ABS)	88.38	95.40	90.71	81.30	88.62	83.29
	Unknown-K (REL)	90.12	87.45	87.30	78.51	77.16	75.50
Tensorflow	Known K	94.62	94.62	94.62	92.06	92.06	92.06
	Unknown-K (ABS)	87.94	95.70	90.61	82.51	95.07	87.27
	Unknown-K (REL)	89.78	82.09	84.00	84.41	88.15	85.03
Pytorch	Known K	90.19	90.19	90.19	94.78	94.78	94.78
	Unknown-K (ABS)	87.52	94.85	89.89	89.89	95.17	91.62
	Unknown-K (REL)	86.55	89.36	87.09	90.63	86.75	87.00

ing a LambdaMART objective, the model learns complex feature interactions while emphasizing rank quality rather than absolute similarity, reducing its dependency on additional semantic channels. Second, the *iterative pick–remove–renormalize* strategy with *Unknown-K* stopping (ABS/REL) continuously recalibrates per-issue scores, further diminishing the marginal utility of transformer-based embeddings. Third, one-to-many issue–commit recovery is inherently a *pattern-matching task*, where lexical embeddings naturally align issue and commit text. In this setting, the combination of TF–IDF and LambdaMART proves especially effective, leaving limited headroom for heavy semantic embeddings.

When can CodeBERT help? Despite limited overall gains, CodeBERT may offer benefits in cases where issue and commit descriptions are sparse, noisy, or paraphrased, or in contexts requiring richer semantic explanations or graph-structured reasoning. In such scenarios, semantic embeddings could complement lexical matching, but for the majority of real-world repositories, lightweight IR features combined with LambdaMART remain both effective and efficient.

Takeaway. Incorporating CodeBERT is optional: while it may yield modest improvements in special cases, our results show that the proposed framework already achieves strong performance without relying on computationally expensive transformer-based embeddings.

RQ4: How do LinkRank and LinkRank-C2I perform in cross-project settings, where models trained on one programming language or repository are evaluated on others?

We performed the cross-project evaluation of our approaches and the baseline models. This was conducted by training on one programming language dataset and testing on others. We considered three settings: training on C++, training

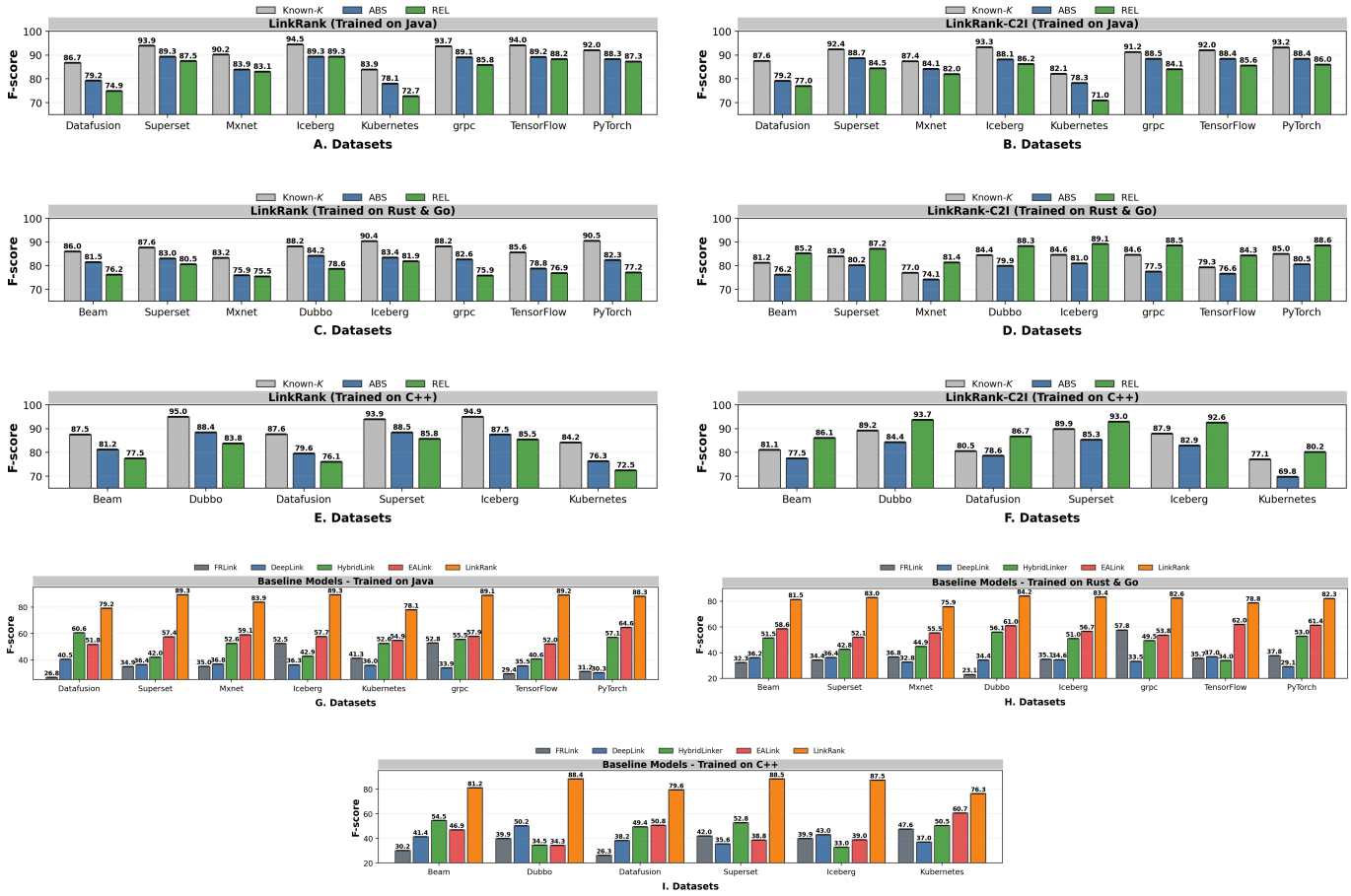

Fig. 3. Cross-project evaluation of LINKRANK, LINKRANK-C2I, and baseline methods. Models are trained on one language family (Java, Go & Rust, or C++) and evaluated on repositories written in other languages. Panels for our approaches show per-dataset F_1 under the Known-K, ABS, and REL regimes; panels for baselines report per-dataset F_1 for FRLink, DeepLink, HybridLinker, and EALink.

on Java, and training on a mixed Rust and Go dataset. All results are presented in the bar plots of Figure 3.

For our proposed approaches, we observe a moderate performance drop compared to the non-cross-project setting. For LinkRank, training on Rust and Go leads to an approximate 5% decrease in the ABS setting and around 7% in the REL setting, highlighting the sensitivity of the model to language differences. For LinkRank-C2I, the decline is more pronounced, with about a 10% drop under the Known-K scenario and roughly 6% in the ABS setting. Despite these decreases, both models continue to achieve competitive scores across repositories, showing that our learning-to-rank framework remains effective and robust even in challenging cross-project scenarios.

The baseline results can also be seen in the bar plots. When we compute the average performance drop across all test datasets, FRLink shows the largest degradation, 19.24% when trained on Rust and Go, and around 18% when trained on either C++ or Java. DeepLink drops by 14.27% (Rust and Go), about 8% (C++), and 13.51% (Java). HybridLinker exhibits 10.9% (Rust and Go), 13.11% (C++), and around 8% (Java). EALink is comparatively more stable, with about 4% (Rust and Go), 16.4% (C++), and 5.47% (Java). Taken together, the baselines experience non-trivial but heterogeneous losses under cross-project transfer, with EALink generally the most resilient and FRLink the most sensitive. Overall, we see that LinkRank remains comparatively more stable and effective across language boundaries than the baseline methods.

RQ5: What are the training and inference costs of the proposed models compared to baselines?

We conducted a comparative analysis of training and testing times across our approaches and the baseline models. Figure 4 presents efficiency (x-axis, in minutes, log scale) together with effectiveness (y-axis, F-score). The results show that our LinkRank variants achieve strong predictive performance while remaining significantly more lightweight. For instance, LinkRank and LinkRank-C2I complete training within 30–40 minutes and testing within 5–7 minutes, yet yield F-scores above 83. In contrast, baselines such as DeepLink and EALink consume far more computational resources (160–225 minutes of training and 10–15 minutes of inference) while producing substantially lower accuracy.

A key observation is that adding CodeBERT embeddings increases both training and testing costs by 3–4× (e.g., 95–110 minutes of training), but delivers only marginal improvements in F-score. This highlights that the core design of LinkRank is already well suited for the largely pattern-matching nature of issue–commit recovery.

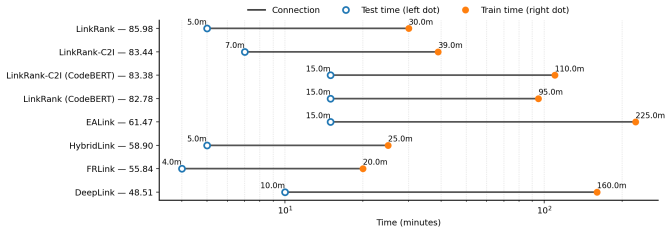

Fig. 4. Training and inference time comparison of LinkRank, LinkRank-C2I, and baseline models. Each line connects test (left dot) and train (right dot) times, while the y-axis annotates the corresponding F-scores. This visualization highlights the balance between computational cost and predictive performance.

VI. Discussion (Limitations & Future Work)

To further validate our framework, we conducted an additional ablation study to assess the importance of our proposed iterative *pick–remove–renormalize* algorithm. In this setting, we replaced the iterative selection with a simple top-K retrieval strategy, directly choosing the highest-scoring commits for each issue without iterative recalibration. We evaluated this under both the Known-K and Unknown-K regimes. As expected, the simplified approach yielded noticeably lower precision and F-scores, confirming that iterative refinement plays a crucial role in improving link recovery accuracy. These ablation results, available in our public repository, reinforce the effectiveness of our iterative algorithm in handling varying issue–commit distributions.

Despite these encouraging outcomes, our study is not without limitations. A primary constraint lies in the size of the proposed dataset. Unlike one-to-one issue–commit links, which are relatively abundant in software repositories, one-to-many links occur far less frequently. As a result, the dataset we curated from GitHub is comparatively small. To safeguard its quality, we conducted manual validation of extracted pairs, carefully checking their correctness and reliability. Nevertheless, it is worth emphasizing that small datasets are not unusual in this research space. Prior efforts on one-to-one traceability, such as Deep-Semi [40] and MPLinker [31], also operated with limited-scale datasets, yet they provided valuable insights that significantly influenced subsequent work. In a similar vein, we position our dataset as a first step toward enabling systematic exploration of one-to-many recovery.

Another consideration relates to the scope of the dataset. Our construction draws exclusively from open-source GitHub projects. This design choice offers transparency, reproducibility, and alignment with the practices of much of the prior literature. However, it also leaves open the question of how well our findings generalize to closed-source, industrial environments, where development practices, commit habits, and issue tracking workflows may differ. To bridge this gap, we are actively exploring collaborations with industry partners to access proprietary datasets and extend our evaluation to more diverse settings. We view this as an important next step, but also note that open-source repositories remain the de facto benchmark for the majority of existing research in this field.

Finally, while our manual validation process helped to reduce errors, the possibility of residual noise cannot be completely eliminated. Mining software repositories at scale inevitably introduces challenges: commit messages may be ambiguous, issue descriptions incomplete, and linking conventions inconsistent across projects. We recognize this as an internal threat to validity. In future work, this can be further mitigated by engaging multiple annotators for cross-checking, quantifying inter-rater agreement, and incorporating semi-automated verification pipelines to systematically filter potential noise.

Taken together, these observations highlight both the promise and the boundaries of our contribution. By acknowledging dataset size, scope, and potential noise, we aim to provide a transparent account of our study while charting clear directions for improvement. Despite these limitations, we believe our dataset and findings offer a timely and concrete foundation for advancing the study of one-to-many issue–commit traceability.

VII. Conclusion

In this paper, we introduced LinkRank, a learning-to-rank framework for recovering one-to-many issue–commit links, a setting often encountered in practice but overlooked by prior one-to-one approaches. To enable this study, we constructed and validated a novel dataset spanning multiple programming languages. Our experiments showed that LinkRank and its variant LinkRank-C2I consistently outperform baselines under both Known-K and Unknown-K regimes, while maintaining competitive performance in cross-project evaluations, thus demonstrating robustness across language boundaries. Furthermore, our efficiency study revealed that LinkRank delivers strong predictive performance with substantially lower training and inference costs compared to heavier baselines, underscoring its practicality for large-scale applications.

Data Availability Statement

The code and data is available in the following repo: LinkRank

References

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

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

[3] T. W. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 14–24.

[4] L. Rierson, *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press, 2013.

[5] M. C. Panis, "Successful deployment of requirements traceability in a commercial engineering organization... really," in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 303–307.

[6] 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 *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 51–62.

[7] C. Zhang, Y. Wang, Z. Wei, Y. Xu, J. Wang, H. Li, and R. Ji, "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, 2023, pp. 217–229.

[8] Z. Alshara, A. Shatnawi, H. Eyal-Salman, A.-D. Seriai, and M. Shatnawi, "Pi-link: A ground-truth dataset of links between pull-requests and issues in github," *IEEE Access*, vol. 11, pp. 697–710, 2022.

[9] N. Serrano and I. Ciordia, "Bugzilla, itracker, and other bug trackers," *IEEE software*, vol. 22, no. 2, pp. 11–13, 2005.

[10] T. Sedano, P. Ralph, and C. Péraire, "The product backlog," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 200–211.

[11] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[12] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version control system: A review," *Procedia Computer Science*, vol. 135, pp. 408–415, 2018.

[13] 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 arXiv:2211.00381*, 2022.

[14] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 97–106.

[15] C. Brindescu *et al.*, "How do centralized and distributed version control systems impact software changes, acm, 2014, retrieved online on may 1, 2019," *Retrieved from the internet:.(Year: 2014)*, pp. 322–333.

[16] A. Mahmoud, N. Niu, and S. Xu, "A semantic relatedness approach for traceability link recovery," in *2012 20th IEEE international conference on program comprehension (ICPC)*. IEEE, 2012, pp. 183–192.

[17] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 15–25.

[18] A. Bachmann and A. Bernstein, "Software process data quality and characteristics: a historical view on open and closed source projects," in *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, 2009, pp. 119–128.

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

[20] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. Gall, "Discovering loners and phantoms in commit and issue data," in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 4–14.

[21] P. R. Mazrae, M. Izadi, and A. Heydarnoori, "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, 2021, pp. 263–273.

[22] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 834–845.

[23] Y. Sun, C. Chen, Q. Wang, and B. Boehm, "Improving missing issue-commit link recovery using positive and unlabeled data," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 147–152.

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

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

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

[27] 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 *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 434–444.

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

[29] J. Lan, L. Gong, J. Zhang, and H. Zhang, "Btlink: automatic link recovery between issues and commits based on pre-trained bert model," *Empirical Software Engineering*, vol. 28, no. 4, pp. 1–55, 2023.

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

[31] B. Wang, Y. Deng, R. Luo, P. Liang, and T. Bi, "Mplinker: Multi-template prompt-tuning with adversarial training for issue-commit link recovery," *Journal of Systems and Software*, p. 112351, 2025.

[32] C. Bird, A. Bachmann, F. Rahman, and A. Bernstein, "Linkster: enabling efficient manual inspection and annotation of mined data," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 369–370.

[33] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2014, pp. 275–284.

[34] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 709–712.

[35] GitHub, "Rate limits for the rest api," 2022, accessed: February 7, 2025. [Online]. Available: https://docs.github.com/en/rest/using-the-rest-api/rate-limits-for-the-rest-api?apiVersion=2022-11-28

[36] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[37] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[38] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, pp. 23–581, 2010.

[39] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[40] J. Zhu, G. Xiao, Z. Zheng, and Y. Sui, "Deep semi-supervised learning for recovering traceability links between issues and commits," *Journal of Systems and Software*, vol. 216, p. 112109, 2024.