information

Article

Enhancing Traceability Link Recovery with Fine-Grained Query Expansion Analysis

Tao Peng [1], Kun She [1,*], Yimin Shen [2,*], Xiangliang Xu [3] and Yue Yu [1]

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
[2] School of Big Data and Artificial Intelligence, Chengdu Technological University, Chengdu 610031, China
[3] School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
* Correspondence: kun@uestc.edu.cn (K.S.); shenyimin@cdtu.edu.cn (Y.S.)

Abstract: Requirement traceability links are an essential part of requirement management software and are a basic prerequisite for software artifact changes. The manual establishment of requirement traceability links is time-consuming. When faced with large projects, requirement managers spend a lot of time in establishing relationships from numerous requirements and codes. However, existing techniques for automatic requirement traceability link recovery are limited by the semantic disparity between natural language and programming language, resulting in many methods being less accurate. In this paper, we propose a fine-grained requirement-code traceability link recovery approach based on query expansion, which analyzes the semantic similarity between requirements and codes from a fine-grained perspective, and uses a query expansion technique to establish valid links that deviate from the query, so as to further improve the accuracy of traceability link recovery. Experiments showed that the approach proposed in this paper outperforms state-of-the-art unsupervised traceability link recovery methods, not only specifying the obvious advantages of fine-grained structure analysis for word embedding-based traceability link recovery, but also improving the accuracy of establishing requirement traceability links. The experimental results demonstrate the superiority of our approach.

Keywords: traceability link recovery; requirements engineering; word embeddings; query expansion; natural language processing

check for updates

Citation: Peng, T.; She, K.; Shen, Y.; Xu, X.; Yu, Y. Enhancing Traceability Link Recovery with Fine-Grained Query Expansion Analysis. *Information* **2023**, *14*, 270. https://doi.org/10.3390/info14050270

Academic Editors: Sanjay Misra, Robertas Damaševičius and Bharti Suri

Received: 11 March 2023
Revised: 25 April 2023
Accepted: 28 April 2023
Published: 2 May 2023

1. Introduction

The requirement traceability link is important in software development, maintenance and upgrading. Gotel defines requirement traceability as “the ability to describe and track the lifecycle of a requirement, both forward and backward (i.e., from its origin, through its development and specification, to its subsequent development and use, and through any of these phrases in a period of continuous refinement and iteration)” [1]. The relationship between requirements can be maintained more efficiently by establishing requirement traceability links. In software projects, there is a close relationship between codes and requirements, and when requirements change, the codes should also change accordingly, which is important for acceptance testing and regulatory review. However, creating and maintaining traceability links between requirements and codes is considered a great challenge in software traceability [2]. Manual establishment of requirement traceability links is error-prone and time-consuming.

A large number of methods currently use information retrieval (IR) techniques to solve the traceability link recovery problem. Common IR methods include spatial vector modeling (VSM) [3,4], latent semantic analysis (LSI) [5], and latent Dirichlet allocation (LDA) [6]. In recent years, with the development of computer technology, machine learning (ML) methods and deep learning (DL) methods have also been used for traceability link recovery tasks [7–9]. Most of the current software documents are written in natural language and the existence of requirement traceability links between two artifacts depends heavily on the similarity of the context of the two texts [10]. Traditional IR methods are based on bag-of-words models and rely more on syntactic features of words, so, if two software artifacts with traceability links use different syntax or contain synonyms, it is difficult for IR methods to determine the validity of traceability information [11]. However, most DL methods use an end-to-end training approach, taking the whole requirement document and source code as input, and ignoring important structural information inside the requirement document and source code. At the same time, DL methods rely heavily on large data, yet there are few publicly available datasets in the field of requirement link recovery that are not difficult to fit using DL methods. These problems are the main reasons for the poor performance of DL methods in traceability link recovery tasks.

In order to solve this problem, we propose a fine-grained requirement-code traceability link recovery approach, based on query expansion. This approach divides the requirement artifacts and code artifacts into different semantic elements, according to their structures, and uses word embedding techniques to obtain their sets of word embeddings. Then, the similarity of different elements is obtained by means of similarity functions, and ranked. Query expansion is used to perform secondary queries to obtain more accurate results. Compared with traditional approaches, and related approaches proposed in recent years, our approach obtains a higher F_1 score, and can effectively improve the effect of requirement traceability link recovery by combining fine-grained relevance analysis and query expansion. The contributions of this paper are as follows:

- A novel approach combining fine-grained correlation analysis and query expansion to solve requirement traceability link recovery is proposed.
- The effectiveness of query expansion for improving the accuracy of requirement traceability link recovery tasks is analyzed by means of the experimental results.
- The experimental results show that our approach achieves state-of-the-art results, demonstrating the effectiveness of this approach.

The rest of the paper is organized as follows. Section 2 presents some related work in this field. Section 3 describes our approach and the overall processing flow. Section 4 presents our experimental design and information about the datasets. Section 5 presents the experimental results and analyzes the results. Section 6 describes the validity threats to our approach. Section 7 summarizes the paper and introduces future work.

2. Related Work

Research on requirement traceability link recovery has been going on since the last century. As the research progresses, various approaches have been proposed. From the traditional IR methods to the latest DL methods, many researchers have worked on improving the recall and accuracy of requirement traceability link recovery. The IR methods are commonly used in the field of requirement traceability link recovery and are also applied to some related fields, such as bug location [12]. Traceability link recovery using the IR method has the following main steps: indexing, retrieval, and presentation [11]. Indexing means transforming the original artifact to be retrieved into a data index format that can be processed by IR models. Retrieval means retrieving the target artifact by means of IR models (e.g., VSM, LSI, LDA). Presentation means presenting the results to the requirement managers and providing follow-up operations through visualization and other tools. However, most traditional IR methods are based on bag-of-words (BOW) models, which cannot capture the deep semantic features of textual requirements, and, thus, these methods suffer from poor results on most datasets.

In addition to IR methods, probabilistic modeling approaches are also often used in the study of requirement traceability link recovery, such as the COMET method [13]. The COMET model proposes a hierarchical Bayesian network-based traceability link prediction model that infers the probability of the existence of a traceability link between given artifacts by combining information from several different measures of text similarity through a constructed hierarchical Bayesian network model.

With the rise of deep learning in recent years, researchers have also applied this latest technique to the field of requirement traceability link recovery. Deep learning uses large datasets to train models that enable them to understand the deep semantic meaning of textual artifacts and, thereby, to improve the effectiveness of traceability link building. Guo [7] proposed a deep learning-based model for the traceability link generation task using word embedding techniques and recurrent neural network (RNN) methods. Guo obtained word embedding models by pre-training them on a large dataset and used the obtained word embedding models to generate a high-dimensional word vector to capture the distributional semantic representation and co-occurrence word statistics of each word. Afterwards, Guo trained the RNN model by using a dataset from the PCT (Positive Train Control) domain. Finally, the trained model was used to obtain a vector representing the semantic information of the artifacts, which determined whether there was a traceability relationship by comparing the similarity of the two vectors. However, DL methods rely on a large number of datasets for training, and it is difficult to obtain enough datasets for training in many scenarios, which, ultimately, leads to poor practicability.

One of the important reasons for the application of these methods is to explore the deep semantic features of software artifacts and bridge the semantic gap between them.

Some approaches seek to further improve the effectiveness of traceability link recovery by means of fine-grained analysis strategies. [14] proposed a fine-grained requirements-code traceability link recovery method, which divides requirements into different elements, according to the use case template, while extracting the important elements (class names, method names, invocation relationships) from the codes. The software artifact elements are employed as word embedding vectors for similarity comparison. Finally, a kind of "voting" approach is used to obtain the recovered traceability links. This method was compared with several state-of-the-art methods and showed good performance, indicating that fine-grained analysis of software artifact elements assists the recovery of traceability link relationships.

Fine-grained analysis has been used in many studies as an important method for traceability link recovery. However, fine-grain-based requirement traceability link recovery only relies on the similarity between elements and neglects to consider the task of constructing relationships between elements as a retrieval task. This means that many retrieval gain techniques are not used in the traceability link recovery task. In contrast, this paper takes this into account and uses the query expansion technique to correct the elements of the query when establishing the relationships between fine-grained elements, and, then, uses secondary retrieval to further enhance the effectiveness of traceability link recovery.

3. Our Approach

We propose a **F**ine-grained **Q**uery **E**xpansion **T**raceability **L**ink **R**ecovery (FQETLR) approach. Our approach analyzes requirements and codes at a fine-grained level and represents textual requirements and programming language codes as sets of vectors by means of word embedding technology. Then, the similarity between each requirement element and the code element is computed, and can be used to adjust the query center, so as to be close to the associated code, by means of query expansion, and, thereby, improve its recovery accuracy. The process of query expansion is shown in Figure 1. For the fixed model parameters, the size of the query range (light green circle) is fixed. However, the query expansion technique ensures the source artifacts (green stars) move closer to the target artifacts (dashed stars), and the target artifact that was ignored (C3) is now retrieved, improving retrieval effectiveness. Compared with traditional IR methods that use only the superficial syntactic similarity of natural language, the use of word embedding techniques can further explore the deep semantic similarity between natural language and programming language to obtain higher matching accuracy. At the same time, our approach does not require a large number of labeled datasets for training and is generalizable across software domains, while ensuring good accuracy.

Our approach establishes traceability links between requirements and codes in the following steps:

(1) Parse the requirements and codes into fine-grained elements, according to their structures, and represent them as vectors through word embeddings.
(2) Calculate the similarity between different elements.
(3) Use the query expansion technique to perform secondary queries to filter the results.
(4) Aggregate the fine-grained relations to obtain traceability links.

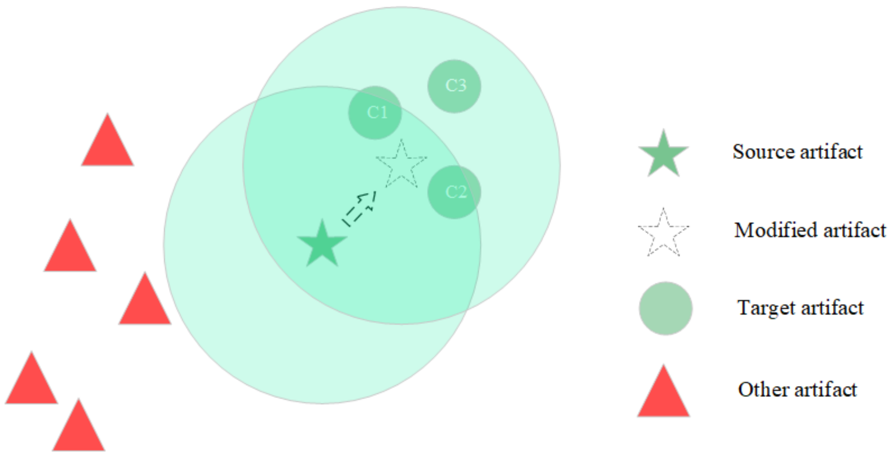


Figure 1. This figure shows how the query expansion method can further improve the effectiveness of traceability link recovery by changing the query center.

3.1. Representation

Software Artifact is often represented as vectors in requirement-code traceability link recovery tasks through word embedding techniques, which allows for better representation of words in the evaluation of requirement links similarity. Some important word embedding methods have been used in many natural language processing tasks, including Word2Vector, Glove [15], and fastText. These word embedding methods have shown great performance in many scenarios. In this paper, we used the fastText model to represent both natural language requirements and programming language codes as word vectors.

We used pre-trained fastText to represent each word of the requirements and codes as word vectors. The training datasets were Wikipedia and CommonCrawl datasets [16], which contain texts from various domains. Using common domain datasets for training enabled the model to better represent different software terminologies and vocabularies.

The raw requirements and codes needed to be pre-processed. The pre-processing steps were similar to those in [14]. For the requirements document, we used Python's NLTK to remove operations for stop words, special words, and unsupported encoding representations in the requirements document, and we used spaCy to lemmatize words. Taking Java as an example of source code files, class and method naming uses camel nomenclature. We needed to split method names into initial lowercase words and to remove some keywords in Java (public int String, etc.), as well as stop words.

Using fine-grained analysis means that important parts of the source code are used for traceability link recovery, rather than the entire source file being analyzed as a whole. The class name, method name, and comment information describe the main functionality of a class, so the token of the source code file needs to be constructed with this information embedded. At the same time, the methods in the class are likely to be duplicated, so the "class name + method name + parameter" was used as the artifact element of the source code. Some of the methods in the class are private methods, which only serve the methods inside the class and do not directly serve the external functional implementation, so the private methods can be ignored.

3.2. Similarity Function

After the embedding representation, the requirement and source code elements are transformed into vector representations. The similarity function is used to identify the similarity between each requirement element and source code element.

Cosine similarity [4] has been widely used as an important similarity metric in many traceability link recovery studies [9,17,18]. Both traditional IR methods and word embedding methods can use cosine similarity to measure the similarity of software artifacts. However, it is not reasonable to calculate the similarity of each word's corresponding vector individually and then to take the maximum or minimum similarity as the similarity of the whole document [14]. This method is too arbitrary and it does not reflect the overall similarity of the vectors corresponding to the two documents well. WMD (Word Mover's Distance) calculates the distance between vector representations of two elements by solving a linear programming problem for the minimum sum of the distance that each word moves from one element to another. Therefore, we used WMD as the similarity function to discriminate the similarity between two software artifacts.

For each vector of software artifacts represented by the fastText model, the distance between the requirement artifact and the code artifact was calculated by means of WMD. A larger WMD value indicated greater distance, and a lower similarity, between two documents. To facilitate the subsequent comparison process, this paper normalized the distance between two software artifacts with a distance value of between [0,1]. It is worth noting that the smaller the distance is, the higher the similarity becomes.

It is necessary to calculate the similarity s_{ij_ij} of a query q_{ij} to each source code and to rank the similarities in descending order so as to find the top k (the exact value of k is set empirically, mainly related to the size of the dataset, the number of query results, etc.). The highest similarities, $\{s_{ij\_i_1j_1}, s_{ij\_i_2j_2}, \ldots, s_{ij\_i_kj_k}\}$, correspond to the source code elements $d_k = \{d_{i_1j_1}, d_{i_2j_2}, \ldots, d_{i_kj_k}\}$. Compute the similarity of each code element d_{ij} and each element of d_k and average to get the similarity offset v_{ij} of the code element d_{ij}, affecting the query q_{ij} with a certain weight φ on the offset. The weight φ was set to 0.1 in this paper. We define the update formula for s'_{ij_ij} as Equation (1):

$$s'_{ij\_ij} = (1-\varphi)s_{ij\_ij} + \varphi v_{i,j} \tag{1}$$

Get the updated s'_{ij_ij} as the modified similarity.

3.3. Aggregation Function

In this paper, requirements and source codes were divided into multiple elements. The elements on the requirements side and the elements on the code side satisfied the traceability relationship in the form of Cartesian product and corresponded to a similarity degree. However, in the actual requirement-code traceability task, a requirement element may not have a code element corresponding to it. Likewise, a code element may not have a requirement element corresponding to it. Therefore, the requirement-code traceability link relationship is established not only by being based on the traceability relationship between individual elements, but by also requiring aggregation functions to perform an aggregation analysis of the results. By analyzing the relationship between different elements of a requirement and different elements of the source code the traceability relationship between the requirement and the source code can be established. The overall architecture is shown in Figure 2. In the requirement traceability link recovery task, traceability links between sources and target artifacts need to be established. First, we extract the elements of requirement artifacts and code artifacts. For requirements, we extract the requirement elements according to the use case template. For codes, we extract the method names, annotations, and call relationship information in the class through syntax analysis. Then, the relationship between requirement elements and code elements is initially established by fine-grained analysis, and we establish element traceability relationships by analyzing the semantic similarity between different elements. Next, we perform secondary queries using query expansion techniques to update the traceability relationships between elements to enhance the retrieval capability of the model. Finally, the relationships between elements are aggregated by an aggregation step through a kind of voting method to obtain the complete traceability links.

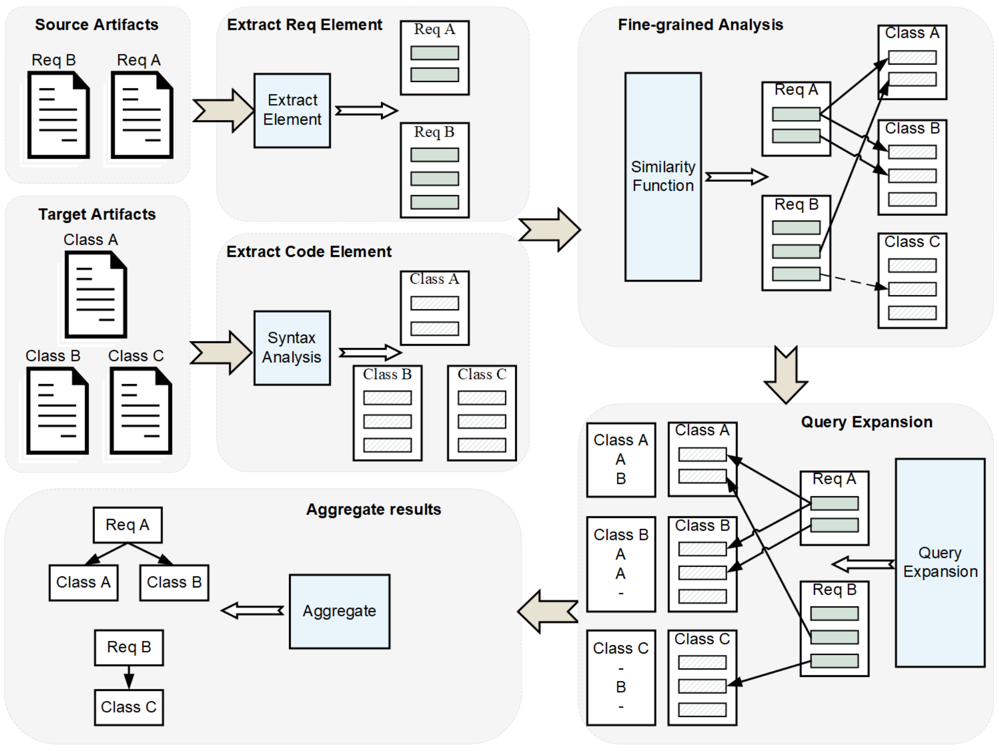


Figure 2. Overview of the FQETLR, showing how it transforms requirement artifacts and code artifacts into fine-grained elements and improves traceability link recovery performance using query expansion.

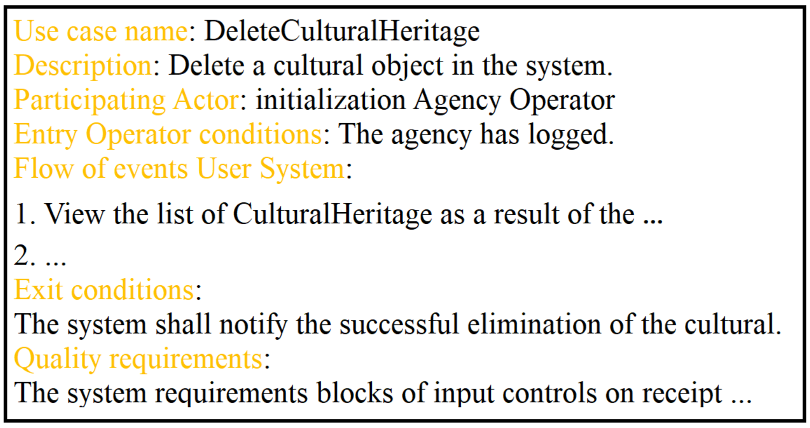
The relationship between requirement and source code depends mainly on the elemental relationships having high similarity. Elemental relationships with low similarity contribute less to the establishment of the traceability relationship. So, we first filter out the elemental pairs that contribute less to the establishment of the traceability link by means of a major threshold, and keep the elemental pairs that contribute more. Then, we use voting to obtain the candidate result. The candidate result indicates whether a result has a high probability of being a valid traceability link. A more rigorous threshold is required to determine the final result. A higher threshold is selected as the final threshold. When there is no less than one candidate result with similarity greater than the final threshold, this indicates that there is a traceability link between the requirement and the source code corresponding to this candidate result. Otherwise, the link should be removed.

Compared to [14], our approach further increases the difference between elements with and without traceability links by performing a secondary query through a query expansion technique. There is less similarity between elements that are originally irrelevant, and more similarity between elements that are relevant. In the aggregation stage, irrelevant candidate links are more easily filtered out by the major threshold and have less impact on the voting process, while traceability links are more easily considered, due to the higher similarity between relevant elements. Our approach makes it easier for the dual thresholds to distinguish valid traceability links.

3.4. Requirement Document Structure

In some software projects, there are structured parts inside the requirements document. However, many approaches ignore the internal structure of the requirements and create traceability links between the entire requirement and the source code or other software artifacts. This ignores the importance of the different parts within the requirement and the differences in descriptive content, which makes the overall similarity between the requirements and their corresponding source codes and other artifacts low, resulting in unsatisfactory final results.

The eTour dataset was taken as an example. This dataset comes from the Center of Excellence for Software & Systems Traceability (CoEST) [19], and the internal structure of its requirements is shown in Figure 3. Requirements usually contain parts, such as use case name, description, participant and workflow, which are obviously not consistent in terms of their presentation. Our approach performs a fine-grained analysis of the requirements, extracts the features of different domains of the requirements as different elements, and performs similarity analysis with different elements of the source code to better recover the traceability link.


Use case name: DeleteCulturalHeritage
Description: Delete a cultural object in the system.
Participating Actor: initialization Agency Operator
Entry Operator conditions: The agency has logged.
Flow of events User System:

1. View the list of CulturalHeritage as a result of the ...

2. ...

Exit conditions:
The system shall notify the successful elimination of the cultural.
Quality requirements:
The system requirements blocks of input controls on receipt ...

Figure 3. Code structure of eTour dataset.

3.5. Code Structure

The structure of the source code is stricter and more standardized (the exact structure varies slightly depending on the language) than that of the structure of the requirement code. Common object-oriented languages (such as Java, Python, C++, etc.) have some common concepts: interface, inheritance, class, method, annotation, etc. These parts are important for establishing traceability links between requirements and source codes.

For a class file, the focus is on the methods of the class file, which form elements of the source code in the form of "class name + method name", with some pre-processing (word splitting, lowercase initials). In the Java language, private methods only serve other methods within the class, and private methods cannot be called from outside the class. Private methods do not contribute to the creation of traceability links, so they are not included as elements [2]. The method's comment, containing detailed descriptive information about the method, is important for establishing traceability links. Some special characters, extra flags, and javadoc are removed from the comments. Java comments carry information about method parameters and return values, which need to be removed as well.

Methods of one class often call methods of another class, and studies have shown that this cross-class call information is useful for improving traceability link recovery [14,20,21]. When a method of one class calls a method of another class, it indicates the existence of a reference relationship between the two classes, which helps in the establishment of a traceability relationship between requirements and codes. A class A calls methods of one or more other related classes B. The similarity between class B and its requirement has an impact on the similarity between class A and its requirement. Based on such an idea, we can integrate the similarity results by using a weighting strategy to weight the similarity between the requirement code and the requirement call code. If there are multiple related classes, we take the mean value of these similarities as the similarity between requirement call codes.

A method element d_{ij} of a source code document is called $cd_{ij} \in CD$, where the weighted value is β. Using the called method, we define the update formula for s'_{ij_ij} as Equation (2):

$$s'_{ij\_ij} = (1-\beta)Similarity(q_{ij},\ d_{ij}) + \beta\frac{\sum_{cd_{ij}\in D} Similarity(cd_{ij},\ q_{ij})}{|CD|} \tag{2}$$

It is worth noting that β was set to 0.1 in this paper. The called method was also used in paper [2], and we set our parameter value to be the same as that in paper [2].

3.6. Example

This subsection describes the traceability link recovery process by means of a specific example.

First, we needed to analyze software artifacts at a fine-grained level, that is, we needed to extract the elements of requirements and codes. The internal content of the requirement file contains the name, description, workflow, etc. The content of the requirement document is shown in Figure 3. We extracted the parts of the requirement into individual elements by means of a requirement extraction template. In detail, we divided the content of a single requirement file into requirement name, requirement description, and requirement workflow. These elements formed an array *['DeleteCulturalHeritage',' Delete a cultural object in the system.',…]*.

In regard to the code file, we used the Java code analysis tool to extract method information (class method and comment) and the call relationship. These elements formed a map *{'Advertisement.clearNews(int)':'Method which removes news…', 'Advertisement.insertNews(BeanNews)':'Insert a new news…',…}*.

Then, these elements were split, and stop words were removed to split the words *[['delete', 'cultural', 'heritage'], …]*.

Since natural language cannot be compared for similarity, we used fastText for word embedding, which converted the elements into vector sets and compared each requirement element with each code element using WMD to obtain a similarity matrix M. Each row represents a requirement element, each column represents a code element, and the corresponding value in the matrix represents the similarity between requirement element and code element.

Considering the call relationship and query expansion, we also needed to correct the similarity between elements using Equations (1) and (2), whereby a secondary query was performed. A new matrix M' was obtained.

Subsequently, we needed to use the voting mechanism to get the final traceability links. For the link pairs $< R1_2, C1_1, 0.2 >, < R2_1, C1_1, 0.8 >, < R1_3, C1_3, 0.7 >, < R2_2, C1_3, 0.7 >$, each link consists of a triple. The value Ri_j denotes the j -th requirement element in the i-th requirement, and Ci_j denotes the j-th code element in the i-th code. The last one denotes the similarity between the two elements, which is cached in the M'.

Voting was done by first filtering out the less similar links (i.e. $< R1_2, C1_1, 0.2 >$) using the primary threshold. Each code contains multiple elements that are candidates for linking to multiple requirement files, and the voting was done by selecting the requirement file with the highest number of links to that code file and establishing a link to that code file. In this example, there was one link between $R1$ and $C1$, and two links between $R2$ and $C1$. Therefore, for code $C1$, we selected $R2$ and $C1$ as candidate links by voting. Finally, considering that some of the code files did not have traceability links, we used the final threshold to determine whether the candidate link between $R2$ and $C1$ was a valid link. If the maximum similarity between $R2$ and $C1$ was greater than the final threshold, the candidate link was a valid link. If not, it was an invalid link.

4. Research Design

To test the performance of the method proposed in this paper, we experimentally answered the following questions:

RQ1: *How do different model parameters affect the experimental results?*

The model contains several parameters. We investigated the effect of different parameter settings on the experimental results.

RQ2: *To what extent does the query expansion method proposed in this paper improve the effectiveness of requirement-code traceability link recovery?*

Previous works have shown that the fine-grained strategy can help improve the effectiveness of traceability link recovery. We compared the effect of models without query expansion and those with query expansion on different datasets.

RQ3: *How does the model proposed in this paper compare with the state-of-the-art approach?*

Combining both query expansion and fine-grained analysis techniques, we investigated an advanced method for traceability link recovery. Therefore, we compared it with some state-of-the-art methods proposed in recent years.

4.1. Datasets

To answer the above questions, we experimentally validated the above three questions with requirement-code correlation datasets. The dataset used in this paper included four sub-datasets from CoEST, which has been used as a benchmark dataset in several studies on requirement traceability link recovery tasks. Since the study of this paper was on the recovery task of requirement traceability links between requirements and codes, four datasets from CoEST, that contain requirement-code traceability links were used: eTour, iTrust, SMOS, and eAnci.

In the field of requirement traceability link recovery, eTour, which contains use cases for a tour navigation system as well as source code class files, is often used. The use case is written in natural language, in English text, and contains use case information, such as use case name, description, participants, and workflow. The source code is written in Java, and the class contains partial javadoc as well as method annotation information. This dataset is suitable for fine-grained traceability link analysis. The other three datasets are also software systems with requirement use cases and source codes, except that the internal natural language part of the eAnci dataset is written in Italian and lacks descriptions, while the other components are similar to the eTour dataset.

In Table 1, the source artifact refers to requirement, test or code, and the target artifact also refers to these software companions. Our paper studied the traceability link between requirements and code. Therefore, the source artifact referred to the requirement, the target artifact referred to the code, and the link referred to the traceability link between requirement and code. The links referred to the number of traceability links between requirements and codes in the dataset, which were created manually by the dataset creator. We can see that the ratios of source and target artifacts were large and the coverage of source artifacts was low in eAnci dataset. The low coverage of source artifacts meant that many requirements did not have corresponding codes. This caused the traceability link recovery method to easily misidentify traceability links.

Table 1. Information about dataset used in the evaluation. The table lists the number of source artifacts, target artifacts and traceability links.

Project	Domain	Language	Source Code	Source Artifacts	Covered Source	Target Artifacts	Covered Target	Links	Source/ Target
eTour	Tourism	EN/IT	JAVA	58	0.983	116	0.767	308	0.50
iTrust	Healthcare	EN	JAVA	131	0.802	226	0.385	286	0.58
SMOS	Education	EN/IT	JAVA	67	1.000	100	0.684	1044	0.67
eAnci	Governance	EN	JAVA	139	0.281	55	1.000	567	2.53

4.2. Methodology

The requirement traceability link recovery task needs some reasonable metrics to evaluate traceability link recovery work. We used the following metrics to evaluate the performance of our approach: accuracy [22], recall [22], F_1 (F-Measure) [23], and MAP (Mean of Average Precision) [2].

In the requirement link recovery task, the retrieved results were as follows: TP, true sample, indicated that the recovered traceability link was an objectively existing traceability link. TN, true negative sample, indicated that the unrecovered traceability link was an objectively non-existent traceability link. FP, false positive sample, indicated that the recovered traceability link was an objectively non-existent traceability link. FN, false negative sample, indicated that the unrecovered link was an objectively existing traceability link. Based on these situations, a number of evaluation metrics could be used to evaluate the predicted results.

The accuracy rate reflects the ratio of correctly recovered traceability links to all recovered traceability links.

$$Pre = \frac{TP}{TP + FP} \tag{3}$$

The recall rate shows the ratio of correctly recovered traceability links to objectively existing traceability links.

$$Rec = \frac{TP}{TP + FN} \tag{4}$$

However, the above evaluation metrics could only be used as reference metrics for traceability link recovery, because these two metrics were too one-sided. Therefore, F_1 could better evaluate the effectiveness of the traceability links as a metric that integrates the accuracy rate and recall rate. The F_1 metric takes into account both accuracy and recall, and a high F_1 score requires both accuracy and recall to be maintained at a relatively high level.

$$F_1 = \frac{2 \times Pre \times Rec}{Pre + Rec} \tag{5}$$

The above three metrics are prey to the problem of single point value limitation. MAP is often used in multi-categorization to reflect the global information of prediction results. MAP defines the average precision of all query results. The formula for calculating the average precision(AP) is:

$$AP = \frac{\sum_{r=1}^{|retrieved|} (precision(r) \times isLink(r))}{|totalLinks|} \tag{6}$$

where $|totalLinks|$ is the number of all objectively existing traceability links; $|retrieved|$ is the number of traceability links retrieved by the traceability link recovery method; $precision(r)$ shows the precision of the top r-ranked traceability links; and$isLink(r)$ is a binary function that determines whether the traceability link ranked at position r is a valid link.

$$MAP = \frac{\sum_{q=1}^{n} AP_q}{n} \tag{7}$$

where n presents the number of requirements, and AP_q denotes the AP of the q-th requirement.

5. Empirical Results

This section shows the results, and provides an analysis, of the experiments. The experimental results using query expansion with different configurations are analyzed, and the effect of parameters on the experimental results explored by changing the parameters. Finally, we compare our approach with state-of-the-art methods.

RQ1: *How do different model parameters affect the experimental results?*

Obviously, different parameters can have an impact on the experimental results. So, this section explores the extent to which changes in the parameters affected the experimental results. To simplify the problem, we only tested configurations containing FQETLR+uct+mc+cd and FTLR+uct+mc+cd. For the threshold parameters, we varied the major and final thresholds in steps of 0.1, while varying the value of k in steps of 1, and recorded the results after parameter optimization. For FTLR, the results before optimization were obtained with the following parameters: major threshold = 0.59, final threshold = 0.44. These parameters for FTLR are recommended in the paper, and FTLR worked better on the four datasets with these parameters. For FQETLR, the result before optimization was obtained with the following parameters: major threshold = 0.59, final threshold = 0.44 and k = 3 (k is a parameter that is unique to our model). Our model parameters were consistent with FTLR, except for k.

From the Table 2, we can see that, compared with the F_1 score before optimization, the F_1 score after parameter optimization improved, with iTrust improving the most significantly and reaching 0.042. Meanwhile, eTour and eAnci improved the least, increasing to 0.014. Interestingly, the difference between the main and final thresholds of the four datasets after optimization was small, the difference between the maximum and minimum of the main threshold was only 0.08, and the difference between the maximum and minimum of the final threshold was only 0.07. The small difference between the thresholds indicated that the optimal threshold fluctuated less between different datasets, reflecting the high generalizability of our approach. Therefore, in practice, the threshold could be set within a certain range to achieve better results, according to the characteristics of a dataset. In regard to another important parameter, namely, the k-value (the number of elements selected that are most similar to the query), we observed that most of the k-values corresponding to the optimal F_1 score for different datasets were concentrated between 1 and 4. So, we set k to the default parameter between 1 and 4. The k-values could also be set according to the dataset to achieve better results. Meanwhile, our approach was better than the FTLR method in F1 score after parameter optimization on all datasets.

In general, different parameters had some effect on the results of FQETLR. However, the differences of the optimal parameters for different datasets were not large, which indicated that there was high generalizability in our approach.

Table 2. On the FQETLR+uct+mc+cd and FTLR+uct+mc+cd variant, the original threshold combination (org) was compared with the optimized configuration (opt) for each project. In addition, the major thresholds, the final thresholds and the k value for each project's optimized configuration are shown.

Project	Method	Precision		Recall		F_1			Threshold		k
		ORG	OPT	ORG	OPT	ORG	OPT	Improve	Maj	Final	
eTour	FTLR	0.405	0.456	0.565	0.516	0.472	0.484	0.012	0.59	0.42	-
	FQETLR	**0.397**	**0.476**	**0.610**	**0.516**	**0.481**	**0.495**	**0.014**	**0.57**	**0.41**	**1**
iTrust	FTLR	0.180	0.231	0.322	0.273	0.231	0.250	0.19	0.54	0.44	-
	FQETLR	**0.171**	**0.256**	**0.357**	**0.294**	**0.232**	**0.274**	**0.042**	**0.54**	**0.42**	**2**
SMOS	FTLR	0.451	0.370	0.288	0.455	0.352	0.408	0.056	0.62	0.48	-
	FQETLR	**0.444**	**0.402**	**0.317**	**0.416**	**0.370**	**0.409**	**0.039**	**0.62**	**0.46**	**4**
eAnci	FTLR	0.294	0.240	0.220	0.282	0.252	0.259	0.007	0.58	0.48	-
	FQETLR	**0.265**	**0.233**	**0.236**	**0.303**	**0.250**	**0.264**	**0.014**	**0.57**	**0.48**	**4**

Bolded text indicates the experimental results of our approach.

RQ2: *To what extent does the query expansion method proposed in this paper improve the effectiveness of requirement code traceability link recovery?*

In [14], it was stated that the fine-grained strategy helps to improve the recovery of traceability links. In the experimental comparison it can easily be observed that the F_1 of the method using fine-grained analysis, compared to the method without fine-grained analysis, improved on all datasets, which proved the effectiveness of the fine-grained approach. On the other hand, in order to verify the impact of query expansion on the performance of traceability link recovery, we compared the experimental results of the methods using query expansion with, and without, query expansion. The parameter settings of FTLR and FQETLR were the same as those mentioned in RQ1 (major threshold = 0.59, final threshold = 0.44 and k = 3). The results are shown in Table 3. Figure 4 shows the precision/recall curves for our approach on four datasets. From the experimental results, it can be seen that the average F1 of the model using query expansion on the four datasets was better than that of the model not using the query expansion method.

Table 3. Comparing the performance of FQETLR and FTLR with different feature choices. The * indicates our approach. Variants that used call dependencies are marked with cd, method annotations are marked with mc, and used case template structures are marked with uct. *avg* denotes the F_1 mean of this feature selection in different datasets.

Approach	**eTour**				**iTrust**				**SMOS**				**eAnci**				*avg*
	Pre	**Rec**	**F_1**	**MAP**	**Pre**	**Rec**	**F_1**	**MAP**	**Pre**	**Rec**	**F_1**	**MAP**	**Pre**	**Rec**	**F_1**	**MAP**	
FTLR	0.287	0.455	0.352	0.330	0.151	0.297	0.200	0.227	0.417	0.140	0.209	0.398	0.215	0.125	0.158	0.142	0.230
FQETLR	**0.286**	**0.510**	**0.366**	**0.328**	**0.157**	**0.336**	**0.214**	**0.23**	**0.423**	**0.180**	**0.253**	**0.409**	**0.242**	**0.178**	**0.205**	**0.143**	**0.260**
+cd	0.307	0.445	0.363	0.339	0.144	0.255	0.184	0.214	0.420	0.138	0.208	0.399	0.226	0.127	0.163	0.142	0.229
***+cd**	**0.286**	**0.506**	**0.366**	**0.339**	**0.155**	**0.297**	**0.204**	**0.216**	**0.419**	**0.172**	**0.244**	**0.410**	**0.249**	**0.176**	**0.206**	**0.144**	**0.255**
+mc	0.282	0.451	0.347	0.275	0.176	0.353	0.235	0.266	0.439	0.155	0.229	0.420	0.227	0.185	0.204	0.148	0.254
***+mc**	**0.277**	**0.503**	**0.358**	**0.281**	**0.168**	**0.381**	**0.233**	**0.271**	**0.438**	**0.193**	**0.268**	**0.432**	**0.228**	**0.247**	**0.237**	**0.148**	**0.274**
+uct	0.398	0.620	0.485	0.523	0.151	0.297	0.200	0.227	0.426	0.277	0.336	0.418	0.306	0.194	0.237	0.146	0.315
***+uct**	**0.390**	**0.646**	**0.487**	**0.531**	**0.157**	**0.336**	**0.214**	**0.230**	**0.422**	**0.313**	**0.360**	**0.425**	**0.294**	**0.213**	**0.247**	**0.146**	**0.327**
+mc+cd	0.273	0.432	0.335	0.277	0.180	0.322	0.231	0.258	0.438	0.145	0.217	0.418	0.242	0.183	0.209	0.149	0.229
***+mc+cd**	**0.278**	**0.497**	**0.356**	**0.284**	**0.171**	**0.357**	**0.232**	**0.260**	**0.445**	**0.181**	**0.257**	**0.432**	**0.226**	**0.238**	**0.232**	**0.149**	**0.269**
+uct+cd	0.411	0.623	0.495	0.516	0.144	0.255	0.184	0.214	0.439	0.277	0.340	0.421	0.314	0.194	0.240	0.146	0.315
***+uct+cd**	**0.402**	**0.649**	**0.497**	**0.518**	**0.155**	**0.297**	**0.204**	**0.216**	**0.426**	**0.307**	**0.357**	**0.426**	**0.305**	**0.212**	**0.250**	**0.145**	**0.327**
+uct+mc	0.390	0.568	0.462	0.477	0.176	0.353	0.235	0.266	0.443	0.297	0.356	0.442	0.270	0.215	0.239	0.150	0.323
***+uct+mc**	**0.383**	**0.61**	**0.471**	**0.469**	**0.168**	**0.381**	**0.233**	**0.271**	**0.437**	**0.327**	**0.374**	**0.451**	**0.260**	**0.240**	**0.250**	**0.148**	**0.332**
+uct+mc+cd	0.405	0.565	0.472	0.471	0.180	0.322	0.231	0.258	0.451	0.288	0.352	0.442	0.294	0.220	0.252	0.150	0.327
***+uct+mc+cd**	**0.397**	**0.610**	**0.481**	**0.472**	**0.171**	**0.357**	**0.232**	**0.260**	**0.444**	**0.317**	**0.370**	**0.451**	**0.265**	**0.236**	**0.250**	**0.150**	**0.333**

Bolded text indicates the experimental results of our approach.

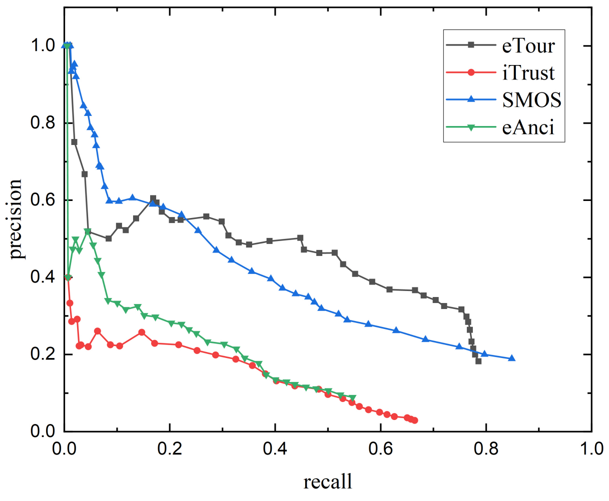


Figure 4. Precision/recall curves on four datasets by varying the final threshold in steps of 0.01 between 0 and 1 for FQETLR+uct+mc+cd.

By analyzing the experimental results, we found that the query expansion method exhibited different effects for different datasets. The query expansion method improved the

F_1 score on the eTour, iTrust, SMOS, and eAnci datasets, by using additional information. However, for the eAnci dataset, using query expansion in the configuration of uct+mc+cd was slightly less effective than the comparison method without the query expansion, which was related to the amount of data in the source and target artifacts. For the eTour, iTrust, and SMOS datasets, the source artifact data size was less than, or approximately equal to, the target artifact data size. However, the number of source artifacts in the eAnci dataset was twice that of the number of target artifacts, while the coverage of source artifacts was also relatively low. This meant that there might have been a large number of source artifacts in the eAnci dataset that had no, or few, target artifacts corresponding to them, and the use of query expansion would result in a large number of invalid links being mistaken for valid links in this case. Thus, the F_1 metric on the eAnci dataset was slightly lower than that of the comparison method. For the iTrust dataset, FQETLR performed only slightly better than FTLR. because its requirement file contained only descriptive content, on which it was difficult to perform fine-grained analysis. This was the reason why the F_1 metric significantly improved by using query expansion on the SMOS and eTour datasets. The more targets a single artifact corresponds to, the more effective the query expansion is.

For datasets (eTour and SMOS) with conditions suitable for using query expansion (where the source artifacts had good coverage and could be analyzed using fine-grained analysis), the query expansion approach performed better on the eTour and SMOS datasets, compared to the approach that not using query expansion.

RQ3: *How does the model proposed in this paper compare with state-of-the-art approach?*

We compare our approach with six recent state-of-the-art methods. The state-of-the-art methods we selected are S2Trace [24], COMET [13], FTLR [14], WQI [17], ALCATRAL [25], and TRAIL [26], of which the first three are unsupervised methods and the last three are supervised methods. We selected these methods with the following considerations: first, these above methods are all proposed in recent years and represent more cutting-edge research in the field of requirement traceability link recovery, which is why we did not select some traditional IR methods (e.g., VSM, LSI, LDA, etc.). Second, these methods are more effective and a comparison of our approach with these methods better illustrates the effectiveness of our approach for establishing requirement traceability link recovery. Finally, after investigating many previous research results, although some of them also show amazing results, the source code of the paper is not publicly available, so it is difficult to compare without a common test dataset.

Table 4 shows the results of our experiments. We chose S2Trace, COMET and FTLR as the comparative unsupervised methods, the reason being that S2Trace also uses embedding learning, while COMET performs well in unseen items. FTLR was the comparison method. For COMET, we accessed the resulting ranking list. Therefore, we were able to additionally calculate the best F1 score and MAP. For FTLR, the configuration of +uct+mc+cd was selected. We used the parameters recommended in the following papers. For the supervised methods, we selected WQI, ALCATRAL and TRAIL, and we used the same parameters as in [14]. For TRAIL, this was calculated by performing 50 random split runs with 90% of the gold standard training data. For ALCATRAL we used a 10% training gold standard because it was the closest to the unsupervised setting. It is not fair to compare unsupervised methods with supervised methods, so this comparison method was only used to measure the effectiveness of our approach.

Table 4. Comparison of FQETLR with state-of-the-art methods. Contrast methods included supervised and unsupervised methods.

Approach	eTour				iTrust				SMOS				eAnci			
	Pre	**Rec**	**F_1**	**MAP**	**Pre**	**Rec**	**F_1**	**MAP**	**Pre**	**Rec**	**F_1**	**MAP**	**Pre**	**Rec**	**F_1**	**MAP**
S2Trace	0.101	0.364	0.158	-	0.196	0.417	0.267	-	-	-	-	-	-	-	-	-
COMET	0.412	0.464	0.437	0.467	0.361	0.231	0.282	0.252	0.166	0.816	0.276	0.293	-	-	-	-
FTLR	0.405	0.565	0.472	0.471	0.18	0.322	0.231	0.258	0.451	0.288	0.352	0.442	0.294	0.220	0.252	0.150
FQETLR	**0.397**	**0.610**	**0.481**	**0.472**	**0.171**	**0.357**	**0.232**	**0.260**	**0.444**	**0.317**	**0.370**	**0.451**	**0.265**	**0.236**	**0.250**	**0.150**
WQI	0.088	0.415	0.145	-	0.198	0.322	0.245	-	-	-	-	-	-	-	-	-
ALCATRAL	0.425	0.427	0.425	-	0.504	0.228	0.309	-	0.513	0.444	0.476	-	-	-	-	-
TRAIL	0.572	0.65	0.608	-	0.568	0.658	0.609	-	0.871	0.735	0.797	-	-	-	-	-

Bolded text indicates the experimental results of our approach.

From the experimental results, interestingly, our proposed method was far ahead of the S2Trace method, with an average improvement of 0.144 in F_1 score on the two datasets, and with significant improvement on the eTour, and slightly weaker improvement than the S2Trace method on the iTrust dataset. In regard to the COMET method, our method improved the F_1 score by 0.029. on average, which was again better than the COMET in regard to the eTour and SMOS, but weaker than the COMET method on the iTrust dataset. We compared the FTLR method with our method and found that our method outperformed the FTLR method in general, with an average improvement of 0.0135 in F_1 score on eTour and SMOS datasets (which are better suited to using query expansion), which showed that combining query expansion and fine-grained analysis could achieve better traceability link recovery results. Compared with several state-of-the-art methods proposed in recent years, our approach achieved better results on some higher-quality datasets in unsupervised traceability link recovery.

Compared to several supervised methods, our approach performed significantly better than WQI, with an average F_1 score improvement of 0.162. Our method outperformed ALCATRAL on the eTour dataset, but not on the other two datasets. The TRAIL method, which performed the best on all three datasets, was far better than the other supervised and unsupervised methods. In summary, our approach showed better results than other unsupervised methods and outperformed some supervised methods.

6. Threats to Validity

In this section, we discuss the threats to the effectiveness of our proposed approach.

The main external threats we face concern datasets. Our experiments were conducted on four datasets that did not cover all kinds of traceability link recovery problems. Furthermore, the number of datasets was not large enough to validate the generalization of the model. The model requires work for fine-grained analysis. However, if it is difficult to fine grain the requirement artifacts themselves an analysis cannot be conducted. FTLR also faces the same problem.

As can be seen from the experiments, it was difficult to achieve good results using query expansion techniques for link relationships with traceability that was too sparse. Therefore, if the number of trace links in the datasets is too small, it affects the effectiveness of trace link recovery.

Additionally, we face internal threats. There are multiple parameters in this model, and the settings of the parameters are related to the dataset itself. Finding the best parameters for the dataset is still a problem.

7. Conclusions

In this paper, we propose an approach that combines query expansion and fine-grained analysis to generate traceability links automatically. Considering the structural characteristics of software artifacts, a fine-grained processing strategy is applied to software artifacts. We found that the recovery of traceability links between elements of software artifacts after being fine-grained is essentially a retrieval task, and the use of query expansion helps to solve this problem.

We compared the effects of different combinations of thresholds on the experimental results. Extensive experiments showed that the thresholds of FQETLR have high generalizability to different projects. Compared to the situation before parameter optimization, the maximum difference between the primary and final thresholds was only 0.08. The smaller difference indicated that the FQETLR had better adaptability on different datasets. When using other datasets, we did not even need to adjust the parameters for the model to still perform better.

To verify the effectiveness of query expansion for traceability link recovery, we systematically compared the possible experimental configuration solutions of the model. The experimental results showed that configurations with query extensions for traceability link recovery outperformed the results exhibited by configurations without query extensions.

Finally, our approach achieved better results by comparing FQETLR with recent unsupervised state-of-the-art methods. The experimental results proved the effectiveness of our approach in traceability link recovery.

We will further explore the possibility of adding intrinsic associations between artifacts to enhance the accuracy of the trace link recovery task. For example, knowledge networks between software artifacts are built using knowledge mapping techniques to uncover intrinsic software artifact associations that can be added as additional information to further bridge the differences between different software artifacts. Different search heuristics can be used to help find traceability links between different artifacts [27]. In addition, we will explore generally applicable combinations of thresholds and k-values.

Author Contributions: Conceptualization, T.P.; methodology, T.P.; formal analysis, T.P.; writing—original draft preparation, T.P.; writing—review and editing, X.X. and Y.Y.; visualization, T.P.; supervision, K.S. and Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by 5G-based research and application demonstration of key technologies for a new generation of automatic intelligent terminals, grant no. 2021YFQ0054.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gotel, O.; Finkelstein, C. An analysis of the requirements traceability problem. In Proceedings of the IEEE International Conference on Requirements Engineering, Colorado Springs, CO, USA, 18–22 April 1994; pp. 94–101. [CrossRef]
2. Florez, J.M. Automated Fine-Grained Requirements-to-Code Traceability Link Recovery. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada, 25–31 May 2019; pp. 222–225. [CrossRef]
3. Falessi, D.; Di Penta, M.; Canfora, G.; Cantone, G. Estimating the Number of Remaining Links in Traceability Recovery (Journal-First Abstract). In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; Association for Computing Machinery: New York, NY, USA, 2018; ASE '18, p. 953. [CrossRef]
4. Salton, G.; Wong, A.; Yang, C.S. A Vector Space Model for Automatic Indexing. *Commun. ACM* **1975**, *18*, 613–620. [CrossRef]
5. Marcus, A.; Maletic, J. Recovering documentation-to-source-code traceability links using latent semantic indexing. In Proceedings of the 25th International Conference on Software Engineering, Portland, OR, USA, 3–10 May 2003; pp. 125–135. [CrossRef]
6. Asuncion, H.U.; Asuncion, A.U.; Taylor, R.N. Software Traceability with Topic Modeling. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, Cape Town, South Africa, 1–8 May 2010; Association for Computing Machinery: New York, NY, USA, 2010; ICSE '10, pp. 95–104. [CrossRef]
7. Guo, J.; Cheng, J.; Cleland-Huang, J. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, 20–28 May 2017; pp. 3–14. [CrossRef]
8. Guo, J.; Gibiec, M.; Cleland-Huang, J. Tackling the term-mismatch problem in automated trace retrieval. *Empir. Softw. Eng.* **2017**, *22*, 1103–1142. [CrossRef]
9. Niu, H.; Keivanloo, I.; Zou, Y. Learning to rank code examples for code search engines. *Empir. Softw. Eng.* **2017**, *22*, 259–291. [CrossRef]

10. Aung, T.W.W.; Huo, H.; Sui, Y. A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis. In Proceedings of the 28th International Conference on Program Comprehension, Seoul, Republic of Korea, 13–15 July 2020; Association for Computing Machinery: New York, NY, USA, 2020; ICPC '20, pp. 14–24. [CrossRef]
11. Mahmoud, A.; Niu, N.; Xu, S. A semantic relatedness approach for traceability link recovery. In Proceedings of the 2012 20th IEEE International Conference on Program Comprehension (ICPC), Passau, Germany, 11–13 June 2012; pp. 183–192. [CrossRef]
12. Ye, X.; Bunescu, R.; Liu, C. Learning to Rank Relevant Files for Bug Reports Using Domain Knowledge. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China, 16–21 November 2014; Association for Computing Machinery: New York, NY, USA, 2014; FSE 2014, pp. 689–699. [CrossRef]
13. Moran, K.; Palacio, D.N.; Bernal-Cárdenas, C.; McCrystal, D.; Poshyvanyk, D.; Shenefiel, C.; Johnson, J. Improving the Effectiveness of Traceability Link Recovery Using Hierarchical Bayesian Networks. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; Association for Computing Machinery: New York, NY, USA, 2020; ICSE '20, pp. 873–885. [CrossRef]
14. Hey, T.; Chen, F.; Weigelt, S.; Tichy, W.F. Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations. In Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), Luxembourg, 27 September–1 October 2021; pp. 12–22. [CrossRef]
15. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
16. Grave, E.; Bojanowski, P.; Gupta, P.; Joulin, A.; Mikolov, T. Learning Word Vectors for 157 Languages. *arXiv* **2018**, arXiv:1802.06893. https://doi.org/10.48550/arXiv.1802.06893
17. Zhao, T.; Cao, Q.; Sun, Q. An Improved Approach to Traceability Recovery Based on Word Embeddings. In Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, China, 4–8 December 2017; pp. 81–89. [CrossRef]
18. Lohar, S.; Amornborvornwong, S.; Zisman, A.; Cleland-Huang, J. Improving Trace Accuracy through Data-Driven Configuration and Composition of Tracing Features. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, Saint Petersburg, Russia, 18–26 August 2013; Association for Computing Machinery: New York, NY, USA, 2013; ESEC/FSE 2013, pp. 378–388. [CrossRef]
19. Center of Excellence for Software & Systems Traceability (CoEST)-Datasets. Available online: http://sarec.nd.edu/coest/datasets.html (accessed on 6 October 2022).
20. Panichella, A.; McMillan, C.; Moritz, E.; Palmieri, D.; Oliveto, R.; Poshyvanyk, D.; De Lucia, A. When and How Using Structural Information to Improve IR-Based Traceability Recovery. In Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering, Genova, Italy, 5–8 March 2013; pp. 199–208. [CrossRef]
21. Kuang, H.; Mäder, P.; Hu, H.; Ghabi, A.; Huang, L.; Lü, J.; Egyed, A. Can method data dependencies support the assessment of traceability between requirements and source code? *J. Softw. Evol. Process* **2015**, *27*, 838–866. [CrossRef]
22. Yang, Y.; Xia, X.; Lo, D.; Bi, T.; Grundy, J.; Yang, X. Predictive Models in Software Engineering: Challenges and Opportunities. *ACM Trans. Softw. Eng. Methodol.* **2022**, *31*, 1–72. [CrossRef]
23. Ruan, H.; Chen, B.; Peng, X.; Zhao, W. DeepLink: Recovering issue-commit links based on deep learning. *J. Syst. Softw.* **2019**, *158*, 110406. [CrossRef]
24. Chen, L.; Wang, D.; Wang, J.; Wang, Q. Enhancing Unsupervised Requirements Traceability with Sequential Semantics. In Proceedings of the 2019 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2–5 December 2019; pp. 23–30. [CrossRef]
25. Mills, C.; Escobar-Avila, J.; Bhattacharya, A.; Kondyukov, G.; Chakraborty, S.; Haiduc, S. Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery. In Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 29 September–4 October 2019; pp. 103–113. [CrossRef]
26. Mills, C.; Escobar-Avila, J.; Haiduc, S. Automatic Traceability Maintenance via Machine Learning Classification. In Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 23–29 September 2018; pp. 369–380. [CrossRef]
27. Prause, C.R. Maintaining Fine-Grained Code Metadata Regardless of Moving, Copying and Merging. In Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, Edmonton, AB, Canada, 20–21 September 2009; pp. 109–118. [CrossRef]